ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación :

INGENIERO DE TELECOMUNICACIÓN

Título del proyecto:

HERRAMIENTA DIDÁCTICA PARA EL MAPEO Y ESTUDIO DE RENDIMIENTO DE OBJETOS ODL EN SGBDRs

Marco Moleres Segura

José Javier Astráin Escola

Alberto Córdoba Izaguirre

Pamplona, 15-09-2010

ÍNDICE

1. Introducción

1.1 Introducción

A lo largo de la historia han ido haciendo acto de presencia diferentes paradigmas de programación. Entendemos paradigma [4] de programación como una visión [1], un enfoque [2], un estilo [3], una filosofía para la construcción del software [2]. Los paradigmas de programación más comunes son el imperativo, el declarativo, el estructurado, el orientado a objetos, el funcional y el lógico [5]. Pese a la gran variedad existente y su aparición a lo largo del tiempo sería muy aventurado afirmar que hay paradigmas mejores que otros. Si que podríamos decir que son diferentes: unos son los más idóneos para resolver un tipo de problemas y hay otros que solucionan más fácilmente otros problemas. En la actualidad, el paradigma que goza de una mayor popularidad y es el más utilizado es el orientado a objetos. De manera sencilla podemos definirlo como el paradigma que utiliza objetos y sus interacciones para la construcción del software. Para poder entender esta definición resulta imprescindible conocer que se entiende por un objeto en el mundo de la programación. Un objeto es una entidad que combina su estado o atributos, las acciones o métodos que se le asocian y una identidad [6].Resaltaremos también que este paradigma introduce conceptos que es importante que conozcamos, al menos los principales:

- Abstracción: denota las características esenciales de un objeto sin entrar en los detalles de la implementación [7,8].
- Encapsulamiento: pese a la existencia de diferencias entre distintos autores, podemos decir que el encapsulamiento consiste en la ocultación de las estructuras de datos y los detalles de la realización de un objeto al resto [8].
- Polimorfismo: el comportamiento de una operación puede ser diferente al actuar sobre diferentes objetos [8].
- Herencia: construir objetos a partir de otros objetos.

Pese a la relevancia y el extendido uso de este paradigma de programación, presenta un gran inconveniente en el momento en el que se pretende almacenar sus objetos en bases de datos. Este problema no sería tal si se utilizaran bases de datos orientadas a objetos. Pese al fuerte avance que estas bases han sufrido en los últimos años, no han logrado implantarse a gran escala, ya sea porque todavía presentan ciertos inconvenientes técnicos, por elevados costes a niveles de adaptación o formación, por problemas de rendimiento o por no estar respaldadas por grandes empresas del sector informático. La cuestión es que son las bases de datos relacionales las más extendidas e implantadas. Entendemos como bases de datos relacionales las que, a grandes rasgos, se basan en tablas compuestas de registros (filas) y campos (columnas) y que se pueden vincular entre ellas. Como se puede apreciar el paradigma empleado en la programación es diferente al paradigma de las bases de datos relacionales y los principales conceptos de cada uno no pueden trasladarse directamente al otro, de modo que en el punto donde convergen aparece una desadaptación también llamada impedancia. Esta impedancia nos va a obligar a tomar medidas para poder pasar de un paradigma a otro, o lo que es lo mismo, almacenar

objetos en bases de datos relacionales. Una de las posibles soluciones al problema es añadir una capa que se encargue de aplicar un patrón de mapeo de objetos en las tablas relacionales [9], por ejemplo los introducidos por Wolfgang Keller para la agregación, asociación y herencia [10]. Aunque se han realizado estudios teóricos acerca del rendimiento ofrecido por los distintos patrones de mapeo no se ha comprobado cual es su comportamiento en entornos reales. Pese a lo estudiado y razonado en las estimaciones teóricas realizadas, hay que tener en cuenta las posibles variaciones que pueden introducir los sistemas gestores de bases de datos relacionales (SGBDR) debido a su complejo funcionamiento interno, pudiendo diferir lo esperado de lo obtenido, incluso es posible que los resultados varíen de un SGBDR a otro. No cabe duda el interés que suscitaría el disponer de una herramienta que nos permita comprobar el rendimiento ofrecido por los distintos patrones de mapeo en un entorno de pruebas real empleando los SGBDR más conocidos e implantados. Máxime si esta herramienta lo hace de un modo visual, automatizado, personalizable e integra un módulo de análisis de resultados.

1.2 Objetivos del proyecto

Pese a que el proyecto tiene como objetivo principal el añadir nuevas funcionalidades a una herramienta ya existente y creada en un proyecto fin de carrera anterior, dichas funcionalidades nos abren un nuevo horizonte en el cual vamos a poder extraer interesantes conclusiones acerca de patrones de mapeo de objetos sobre tablas en SGBDR y sobre el propio comportamiento de dichos SGBDR en situaciones muy concretas. De modo que se puede decir que el presente proyecto tiene tres objetivos:

- Mejorar y añadir nuevas funcionalidades a una herramienta proveniente de un proyecto fin de carrera anterior. Uno de los aspectos más destacables a mejorar en la herramienta ya existente es el relacionado con la capacidad de análisis de las tablas y sus relaciones.
  Las nuevas funcionalidades que se pretende añadir son:
  - Capacidad de funcionamiento con varios SGBDRs.
  - Capacidad de realizar mediciones de rendimiento.
  - Plantillas predefinidas para el mapeo de objetos (asociación, agregación y herencia).
  - Realización automática de mediciones de rendimiento de diferentes peticiones dentro de diferentes mapeos.
  - Almacenamiento, carga, visualización y comparación de los resultados obtenidos en las mediciones.
- Realizar y analizar mediciones de consultas sobre diferentes SGBDR.
- Realizar y analizar mediciones de rendimiento de los diferentes patrones de mapeo y en diferentes SGBDRs.

1.3 Estado del arte

Las bases de datos se han convertido en el pilar básico para el almacenamiento y gestión de la información. El volumen de la información con que se trabaja crece día a día haciendo necesarias importantes inversiones en soluciones de almacenamiento, no solo por

la cantidad de datos a manejar sino también por los tiempos de respuesta. Por ello se hace imprescindible para el correcto aprovechamiento de los recursos disponibles el uso de herramientas que permitan analizar el rendimiento de los sistemas gestores. En la actualidad existen numerosas herramientas dedicadas al análisis de rendimiento. En general estas herramientas requieren conocimientos avanzados en sistemas gestores, desde la preparación, el manejo e incluso el análisis de los resultados obtenidos y suelen basarse en baterías de pruebas ya prefijadas que en muchas ocasiones no se adaptan a las necesidades concretas del usuario.

Pese a que el paradigma de programación más extendido y utilizado es el orientado a objetos son los SGBD de tipo relacional los que imperan, de modo que para almacenar los objetos en estos sistemas gestores se ha de recurrir a técnicas de mapeo. Estas técnicas las podemos aplicar nosotros mismos o podemos hacer uso de herramientas de mapeo objeto-relacional que facilitan la labor del programador. Disponemos de un amplio abanico de ellas, unas de las más extendidas son Hibernate, Castor, JDO…, sin embargo aun utilizando estas herramientas, pese a su comodidad y flexibilidad, hemos de seguir definiendo nosotros mismos los mapeos. Los métodos de carga y salvado no van a ser los óptimos, ni se van a ajustar automáticamente a las necesidades concretas del usuario. Por ello, en aplicaciones muy críticas o en puntos donde se desee mantener un control total, obtener el máximo rendimiento en las operaciones más utilizadas por el usuario puede ser interesante realizar todo el proceso de salvado y carga de objetos de una manera totalmente controlada.

1.4 Descripción de la propuesta

Aun que existe una gran variedad de herramientas orientadas a la medición de rendimiento de SGBDR y existen multitud de herramientas de mapeo objeto-relacional sería interesante poder disponer de una herramienta de manejo muy sencillo e intuitivo que permita realizar mediciones de rendimiento sobre diferentes SGBDRs de una manera simple y eficaz. Una vez obtenidos los resultados de las diferentes pruebas poder analizarlos con la misma herramienta mediante gráficas y tablas interactivas en las que se pueda visualizar diferentes resultados estadísticos, así como poder realizar mezclas comparativas con otras sesiones de pruebas y de este modo facilitar la labor de la extracción de conclusiones. La herramienta también ha de tener la capacidad tanto de realizar mediciones de una manera manual como de una manera automática. En el modo manual se mostrará al usuario toda la información necesaria sobre las tablas y sus relaciones y así poder elaborar adecuadamente las consultas. En modo automático se ha de poder prefijar de una manera sencilla, mediante ficheros de texto, todas las consultas. Ya que la herramienta ha de permitir la realización de pruebas automatizadas se pueden crear unas pruebas prefijadas que nos permitan comparar el rendimiento de diferentes patrones de mapeo de objetos en tablas relacionales y el rendimiento que nos ofrecen los distintos SGBDRs.

2. Sistemas gestores de bases de datos

En los siguientes apartados vamos a describir los sistemas gestores a los que la herramienta va a poder conectarse.

Microsoft Access

Microsoft Access es un sistema de gestión de bases de datos (DBMS) para uso personal o de pequeñas organizaciones.
Su funcionamiento esta basado en un motor llamado Microsoft Jet [11] que básicamente permite el desarrollo de aplicaciones formadas por formularios Windows y código VBA (Visual Basic para Aplicaciones). Sin embargo para la realización del proyecto nos hemos limitado a utilizar el motor Microsoft Jet mediante el uso del driver ODBC (Microsoft Access Driver) sin hacer uso propiamente de Microsoft Access.

Entre sus mayores inconvenientes figuran que no es multiplataforma, pues sólo está disponible para sistemas operativos de Microsoft. Su uso es inadecuado para grandes proyectos de software que requieren tiempos de respuesta críticos o muchos accesos simultáneos a la base de datos.

MySQL

Es el gestor de bases de datos relacionales gratuito y de código abierto más conocido y extendido [12], según cifras aportadas por el propio fabricante supera en número a cualquier otra herramienta de bases de datos. Al ser código abierto (bajo licencia GPL) los usuarios pueden modificarla para adaptarla mejor a sus necesidades concretas. Es multiplataforma permitiendo su disponibilidad en gran cantidad de plataformas y sistemas.
(Linux, FreeBSD, NetBSD, Solaris, Microsoft Windows…).

Aunque en un principio MySQL carecía de numerosos elementos que son considerados esenciales en las bases de datos relacionales, tales como integridad referencial y transacciones, atrajo a los desarrolladores de páginas Web con contenido dinámico, por su simplicidad de instalación, manejo y el ser gratuitos, con el paso del tiempo y su extendido uso, aquellos elementos de los que carecía, están siendo incorporados tanto por desarrollos internos, como por desarrolladores de software libre.
Sus principales características son:

- Amplio subconjunto del lenguaje SQL.
- Disponibilidad en gran cantidad de plataformas y sistemas.
- Diferentes opciones de almacenamiento según si se desea velocidad en las operaciones o el mayor número de operaciones disponibles.
- Transacciones y claves foráneas.
- Conectividad segura.
- Replicación.
- Búsqueda e indexación de campos de texto.

Oracle

Oracle es básicamente una herramienta cliente/servidor para la gestión de Bases de Datos. Es un producto vendido a nivel mundial, aunque la gran potencia que tiene y su elevado precio hacen que sólo se vea en empresas grandes y multinacionales, por norma general. En el desarrollo de páginas web pasa lo mismo: como es un sistema muy caro no está tan extendido como otras bases de datos, por ejemplo, Access, MySQL, SQL Server, etc... La Corporación Oracle ofrece este SGDBR como un producto incorporado a la línea de producción. Oracle funciona en ordenadores personales (PC), microcomputadoras, mainframes y computadoras con procesamiento paralelo masivo. Soporta unos 17 idiomas, corre automáticamente en más de 80 arquitectura de hardware y software distinto sin tener la necesidad de cambiar una sola línea de código. Esto es porque más el 80% de los códigos internos de Oracle son iguales a los establecidos en todas las plataformas de sistemas operativos. Se considera a Oracle como uno de los sistemas de bases de datos más completos, destacando su:

- Soporte de transacciones.
- Estabilidad.
- Escalabilidad.
- Es multiplataforma.

PostgreSQL

Se trata de un sistema de gestión de bases de datos objeto-relacional (ORDBMS) basado en el proyecto POSTGRES, de la universidad de Berkeley. El director de este proyecto es el profesor Michael Stonebraker, y fue patrocinado por Defense Advanced Research Projects Agency (DARPA), el Army Research Office (ARO), el National Science Foundation (NSF), y ESL, Inc [13].

PostGreSQL es una derivación libre (OpenSource) de este proyecto, y utiliza el lenguaje SQL92/SQL99, así como otras características que comentaremos más adelante.

Fue el pionero en muchos de los conceptos existentes en el sistema objeto-relacional actual, incluido, más tarde en otros sistemas de gestión comerciales. PostGreSQL es un sistema objeto-relacional, ya que incluye características de la orientación a objetos, como puede ser la herencia, tipos de datos, funciones, restricciones, disparadores, reglas e integridad transaccional. A pesar de esto, PostGreSQL no es un sistema de gestión de bases de datos puramente orientado a objetos.

- Implementación del estándar SQL92/SQL99.
- Soporta distintos tipos de datos: además del soporte para los tipos base, también soporta datos de tipo fecha, monetarios, elementos gráficos, datos sobre redes (MAC, IP ...), cadenas de bits, etc. También permite la creación de tipos propios.
- Incorpora una estructura de datos array.
- Incorpora funciones de diversa índole: manejo de fechas, geométricas, orientadas a operaciones con redes, etc.
- Permite la declaración de funciones propias, así como la definición de disparadores.
- Soporta el uso de índices, reglas y vistas.

- Incluye herencia entre tablas (aunque no entre objetos, ya que no existen), por lo que a este gestor de bases de datos se le incluye entre los gestores objeto-relacionales.
- Permite la gestión de diferentes usuarios, como también los permisos asignados a cada uno de ellos.

2.1 JDBC

JDBC es una especificación de un conjunto de clases y métodos de operación que permiten a cualquier programa Java acceder a sistemas de bases de datos de forma homogénea [14]. Al igual que ODBC [15], la aplicación de Java debe tener acceso a un driver JDBC adecuado. Este driver es el que implementa la funcionalidad de todas las clases de acceso a datos y proporciona la comunicación entre el API JDBC y la base de datos real.

La necesidad de JDBC, a pesar de la existencia de ODBC, viene dada porque ODBC es un interfaz escrito en lenguaje C, al no ser un lenguaje portable, haría que las aplicaciones Java perdiesen la portabilidad. ODBC tiene además el inconveniente de que se ha de instalar manualmente en cada máquina; al contrario que los drivers JDBC, que al estar escritos en Java son automáticamente instalables, portables y seguros evitando así al usuario las molestias y los inconvenientes de una instalación manual. En la figura 2.1.1 podemos observar las capas [16] existentes para la comunicación con los sistemas gestores de bases de datos.

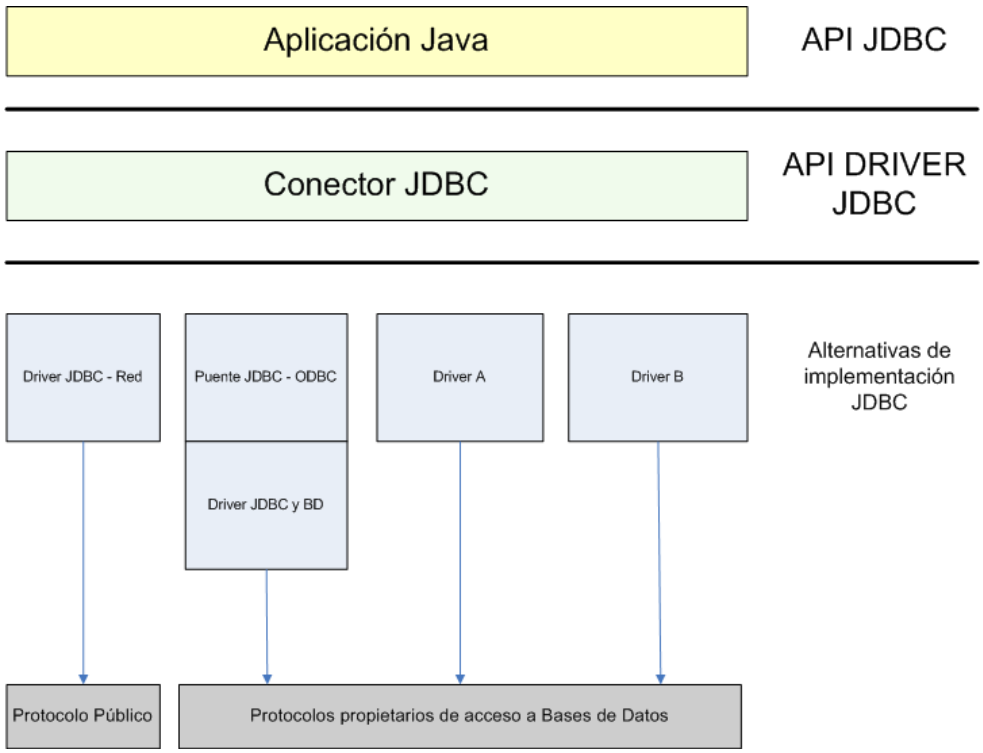


Figura 2.1.1: Capas para la comunicación con SGBD.

Toda la conectividad de bases de datos de Java se basa en sentencias SQL, por lo que se hace imprescindible un conocimiento adecuado de SQL para realizar cualquier clase de operación de bases de datos. Casi todos los entornos de desarrollo Java ofrecen componentes visuales que proporcionan una funcionalidad suficientemente potente sin necesidad de que sea necesario utilizar SQL, aunque para usar directamente el JDK se haga imprescindible. La especificación JDBC requiere que cualquier driver JDBC sea compatible con al menos el nivel «de entrada» de ANSI SQL 92 (ANSI SQL 92 Entry Level).

Acceso de JDBC a Bases de Datos

El API JDBC soporta dos modelos diferentes de acceso a Bases de Datos, los modelos de dos y tres capas.

Modelo de dos capas

Este modelo se basa en que la conexión entre la aplicación Java o el applet que se ejecuta en el navegador, se conectan directamente a la base de datos. En la figura 2.1.2 podemos observar las capas correspondientes a este modelo.

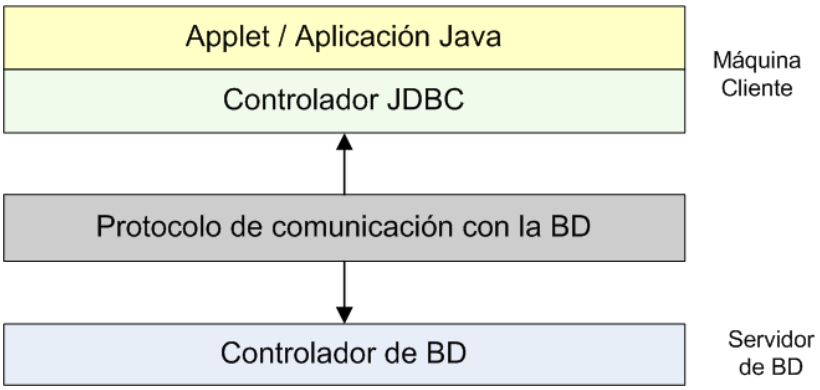


Figura 2.1.2: Modelo JDBC de dos capas.

Esto significa que el driver JDBC específico para conectarse con la base de datos, debe residir en el sistema local. La base de datos puede estar en cualquier otra máquina y se accede a ella mediante la red. Esta es la configuración de típica Cliente/Servidor: el programa cliente envía instrucciones SQL a la base de datos, ésta las procesa y envía los resultados de vuelta a la aplicación.

Modelo de tres capas

En este modelo de acceso a las bases de datos, las instrucciones son enviadas a una capa intermedia entre Cliente y Servidor, que es la que se encarga de enviar las sentencias SQL a la base de datos y recoger el resultado desde la base de datos. En este caso el usuario no tiene contacto directo, ni a través de la red, con la máquina donde reside la base de datos. En la figura 2.1.3 podemos observar la representación de este modelo.

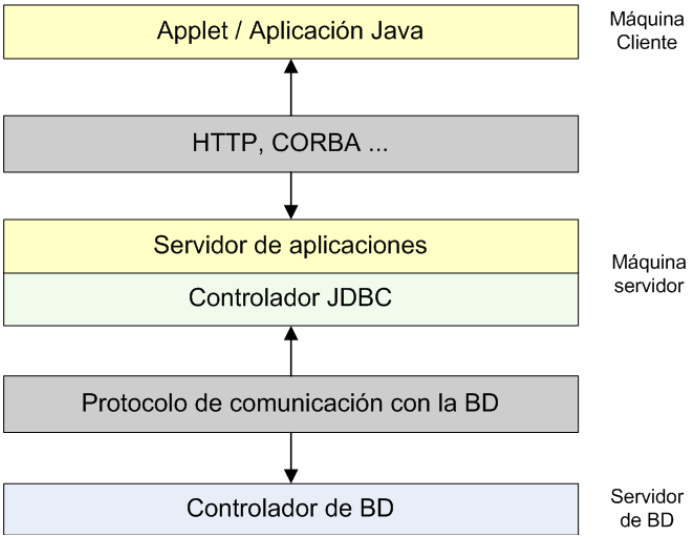


Figura 2.1.3: Modelo JDBC de tres capas.

Este modelo presenta la ventaja de que el nivel intermedio mantiene en todo momento el control del tipo de operaciones que se realizan contra la base de datos, y además, está la ventaja adicional de que los drivers JDBC no tienen que residir en la máquina cliente, lo cual libera al usuario de la instalación de cualquier tipo de driver.

Tipos de drivers

Un driver JDBC puede pertenecer a una de cuatro categorías diferentes en cuanto a la forma de operar.

· Puente JDBC-ODBC

La primera categoría de drivers es la utilizada por Sun inicialmente para popularizar JDBC y consiste en aprovechar todo lo existente, estableciendo un puente entre JDBC y ODBC. Este driver convierte todas las llamadas JDBC a llamadas ODBC y realiza la conversión correspondiente de los resultados. En la figura 2.1.4 se puede observar un diagrama con sus capas.

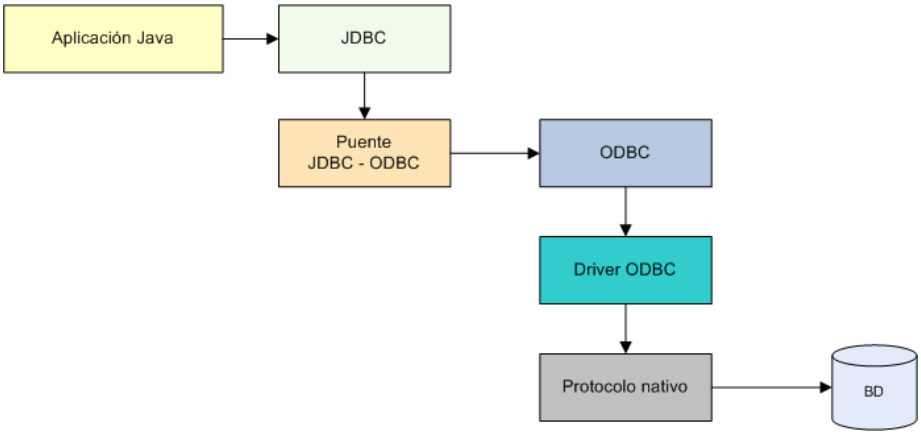


Figura 2.1.4: Capas conexión puente JDBC-ODBC.

La ventaja de este driver, que se proporciona con el JDK, es que Java dispone de acceso inmediato a todas las fuentes posibles de bases de datos y no hay que hacer ninguna configuración adicional aparte de la ya existente. No obstante, tiene dos desventajas muy importantes, por un lado, la mayoría de los drivers ODBC a su vez convierten sus llamadas a llamadas a una librería nativa del fabricante DBMS, con lo cual la lentitud del driver JDBC-ODBC puede ser exasperante, al llevar dos capas adicionales que no añaden funcionalidad alguna; y por otra parte, el puente JDBC-ODBC requiere una instalación ODBC ya existente y configurada.

Lo anterior implica que para distribuir con seguridad una aplicación Java que use JDBC habría que limitarse en primer lugar a entornos Windows (donde está definido ODBC) y en segundo lugar, proporcionar los drivers ODBC adecuados y configurarlos correctamente. Esto hace que este tipo de drivers esté totalmente descartado en el caso de aplicaciones comerciales, e incluso en cualquier otro desarrollo, debe ser considerado como una solución transitoria, porque el desarrollo de drivers totalmente en Java hará innecesario el uso de estos puentes.

· Java/Binario

Este driver se salta la capa ODBC y se conecta directamente con la librería nativa del fabricante del sistema DBMS. Se trata de un driver 100% Java, pero aún así necesita la existencia de un código binario (la librería DBMS) en la máquina del cliente, con las limitaciones y problemas que esto implica. En la figura 2.1.5 esta representadas las capas de este driver.

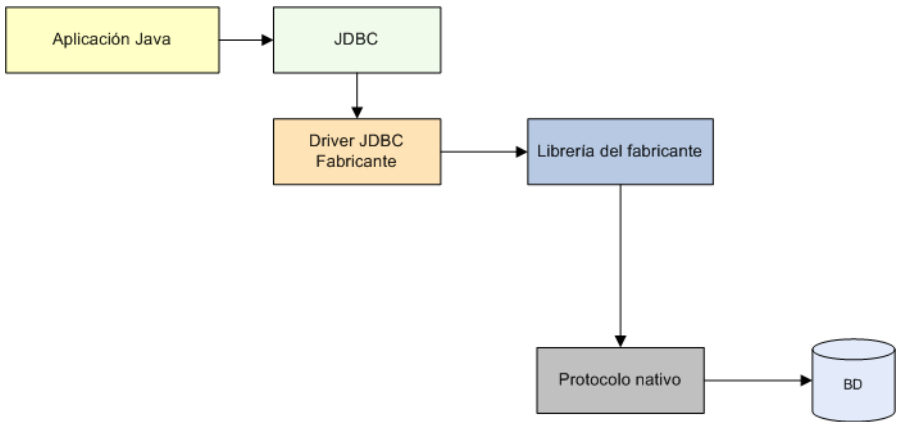


Figura 2.1.5: Capas conexión puente Java/Binario.

· 100% Java/Protocolo nativo

Es un driver realizado completamente en Java que se comunica con el servidor DBMS utilizando el protocolo de red nativo del servidor. Las capas de este tipo de driver están representadas en la figura 2.1.6. De esta forma, el driver no necesita intermediarios para conectarse con el servidor y convierte todas las peticiones JDBC en peticiones de red contra el servidor. La ventaja de este tipo de driver es que es una solución 100% Java y, por lo tanto, independiente de la máquina en la que se va a ejecutar el programa.

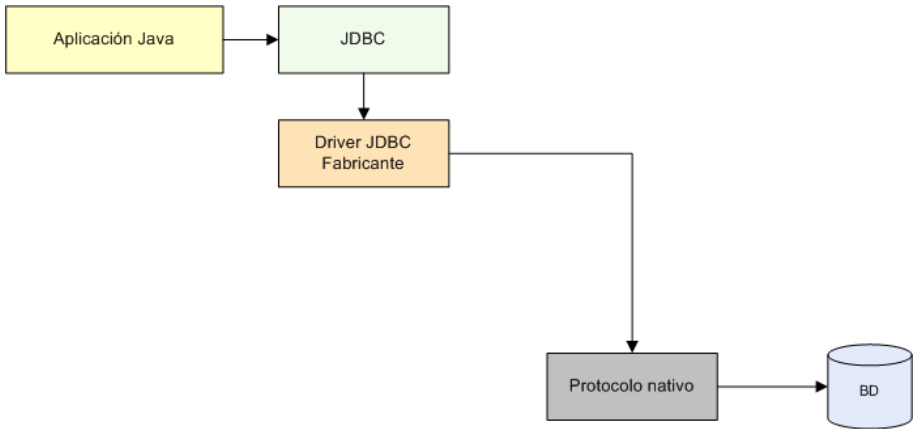


Figura 2.1.6: Capas conexión 100% Java/Protocolo nativo.

La única desventaja de este tipo de drivers es que el cliente está ligado a un servidor DBMS concreto, ya que el protocolo de red que utiliza MS SQL Server por ejemplo no tiene nada que ver con el utilizado por Postgress u Oracle. La mayoría de los fabricantes de bases de datos han incorporado a sus propios drivers JDBC del segundo o tercer tipo, con la ventaja de que no suponen un coste adicional.

· 100% Java/Protocolo independiente

Esta es la opción más flexible, se trata de un driver 100% Java / Protocolo independiente, que requiere la presencia de un intermediario en el servidor. En este caso, el driver JDBC hace las peticiones de datos al intermediario en un protocolo de red independiente del servidor DBMS. El intermediario a su vez, que está ubicado en el lado

del servidor, convierte las peticiones JDBC en peticiones nativas del sistema DBMS. La ventaja de este método es inmediata: el programa que se ejecuta en el cliente, y aparte de las ventajas de los drivers 100% Java, también presenta la independencia respecto al sistema de bases de datos que se encuentra en el servidor.

De esta forma, si una empresa distribuye una aplicación Java para que sus usuarios puedan acceder a su servidor MS SQL y posteriormente decide cambiar el servidor por Oracle, Postgresql o DB2, no necesita volver a distribuir la aplicación, sino que únicamente debe reconfigurar la aplicación residente en el servidor que se encarga de transformar las peticiones de red en peticiones nativas. La única desventaja de este tipo de drivers es que la aplicación intermediaria es una aplicación independiente que suele tener un coste adicional por servidor físico, que hay que añadir al coste del servidor de bases de datos. En la figura 2.1.7 se encuentran representadas las distintas capas correspondientes a este tipo de drivers.

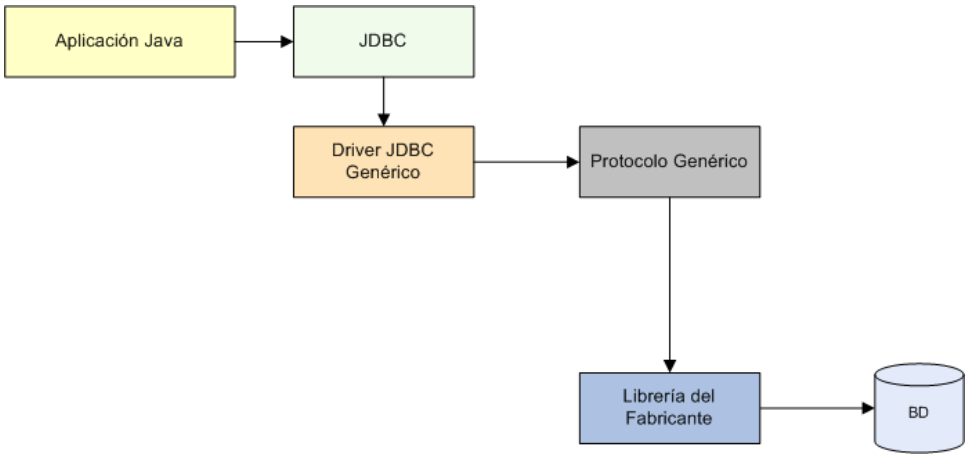


Figura 2.1.7: Capas conexión 100% Java/Protocolo independiente.

.

3. Mapeo de objetos a tablas relacionales

3.1 Introducción

Mapear objetos en tablas es un problema al que ha de enfrentarse toda persona que utiliza un lenguaje de programación orientado a objetos (Java [17], Visual Basic [18], Visual C++ [19]…), pero ha de utilizar bases de datos relacionales (RDBMS) en lugar de bases de datos orientadas a objetos (OODBMS). La orientación a objetos y el modelo relacional son dos paradigmas diferentes de programación, no hay una equivalencia exacta ni única entre ambos conceptos, de modo que crear una correspondencia entre un modelo de objetos y un esquema de relaciones requiere establecer una equivalencia entre distintos conceptos [10].

- La identidad de un objeto y su representación en las tablas.
- Clases y relaciones.
- Relaciones entre las clases y su equivalencia en relaciones entre tablas.
- La herencia sobre tablas.

Estas equivalencias pueden realizarse de distintas formas, pero siempre teniendo en mente que se requiere eficiencia, alto rendimiento, flexibilidad, bajo coste de mantenimiento, etc. Para ello se analizan los siguientes conceptos de la programación orientada a objetos que se van a mapear en bases de datos relacionales.

- Agregación.
- Herencia y polimorfismo.
- Asociación entre clases.

Las consideraciones a tener en cuenta cuando se procede a esto mapeos son:

- *Rendimiento*: Se trata de una de las consideraciones más importantes que se han de tener en cuenta. El modo en que se mapean los objetos en las tablas tiene un gran impacto en el número de accesos que se realizan a la base de datos y dado que la velocidad de discos duros o dispositivos externos es del orden de milisegundos y los ciclos del procesador o de la RAM son del orden de nanosegundos, es importante tener en cuenta al realizar el diseño de las tablas que conviene utilizar ciclos de procesador o RAM antes que realizar accesos a discos duros.
- *Lectura frente a escritura/actualización*: Uno de los problemas que pueden surgir al optar por un tipo de mapeo es el alcanzar un compromiso entre el rendimiento en lectura con el rendimiento en escritura o actualización. Se puede diseñar un mapeo que, por ejemplo, permita realizar la lectura deseada en un solo acceso a la base de datos, pero por el contrario se necesite realizar múltiples accesos para escribir o actualizar objetos; por ello, es muy importante analizar previamente el tipo de acción mayoritaria que se va a realizar sobre la base de datos antes de optar por el tipo de mapeo.

- *Flexibilidad y coste de mantenimiento*: En determinadas situaciones el factor rendimiento deja de ser tan importante como puede ser la flexibilidad del mapeo, dado que puede ser necesario modificar, añadir o eliminar atributos de los objetos, clases o incluso reestructurar la herencia de las clases. Una vez se disponga de un diseño estable de las clases, se puede optar por otro tipo de mapeo que ofrezca un rendimiento óptimo.
- *Rendimiento y redundancia frente a coste de mantenimiento y formas normales*: Mediante el uso de formas normales y factorización se puede eliminar datos redundantes de la base de datos, pero por contra se incrementa el número de accesos a la base de datos necesarios para el manejo de los objetos almacenados, lo cual provoca una pérdida de rendimiento. De modo que el coste de mantenimiento de un modelo de datos entra en conflicto con un buen rendimiento.
- *Espacio consumido frente a rendimiento*: Existen patrones de mapeo que no utilizan más espacio del necesario (campos con valor nulo o similares) y que dejan grandes porciones de datos almacenados sin utilizar. No es sorprendente que los que no realizan un consumo óptimo del espacio sean los que alcancen un mayor rendimiento.
- *Procesamiento de las consultas*: En los sistemas de información existen dos tipos de propósitos que entran en conflicto:
    - Datos preparados para transacciones *online* que tienen que estar listos para ser procesados con un buen rendimiento, en cuyo caso se buscan mapeos con alto rendimiento.
    - Datos con propósito de almacenamiento (*data wharehouse*) que tienen que estar adaptados para peticiones *ad-hoc*. Se buscan mapeos que utilicen formas normales, eviten la redundancia y se factoricen lo máximo posible.

  Como ya se ha visto con anterioridad, se produce un conflicto de intereses debido a que en general el rendimiento se opone a las formas normales y a la factorización. De modo que se ha de estudiar el tipo de los datos que vamos a utilizar.
- *Tipo de aplicación*: Existen determinadas aplicaciones en las que el uso de bases de datos relacionales para el almacenamiento de objetos no es conveniente. Este es el caso de las aplicaciones de tipo CAD, en las que se manejan grandes conjuntos de complejos objetos interrelacionados de modo que las transacciones son muy largas y normalmente sin colisiones, algo para lo que las bases de datos relacionales no han sido diseñadas.

La herramienta generada considera los siguientes patrones de mapeos de objetos a tablas de la agregación, asociación y herencia, siguiendo los patrones de mapeo propuestos por Keller [10], entre distintas clases de objetos. La figura 3.1.1 muestra los distintos patrones de mapeo.

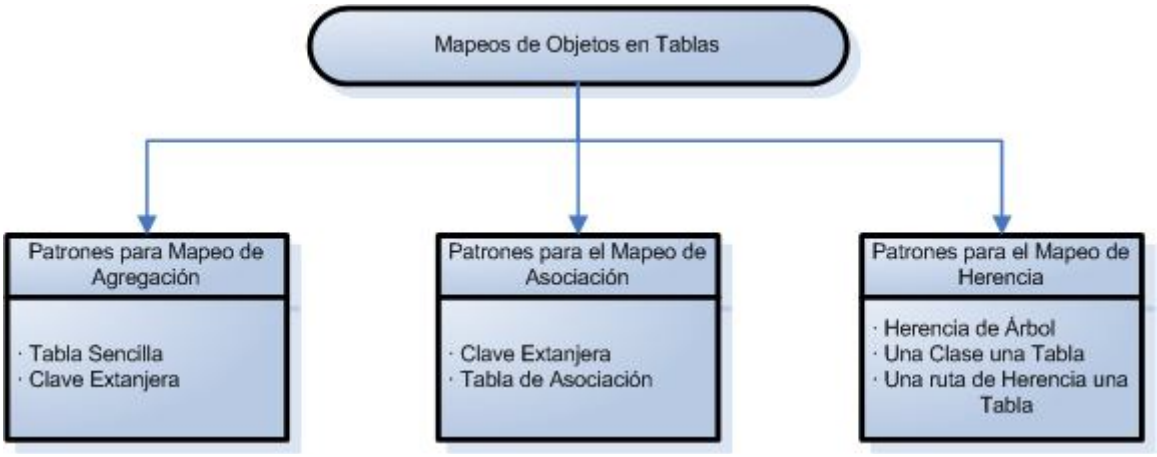


Figura 3.1.1: Patrones de mapeo.

3.2 Agregación

Para mapear la agregación en una base de datos relacional contamos con dos posibles soluciones, mapeo mediante *tabla sencilla* o mapeo mediante *clave extranjera*. Dado el diagrama de clases de la figura 3.2.1, donde *Cliente* es el tipo compuesto o agregado y *Dirección* el tipo componente, el mapeo de esta agregación en tablas relacionales se describe a continuación.

Figura 3.2.1: Agregación.

a) *Mapeo en tabla sencilla*

Consiste en incluir todos los atributos de los objetos componentes en la misma tabla que representa al objeto compuesto. En la figura 3.2.2 se puede observar un ejemplo de mapeo de agregación en tabla sencilla para el ejemplo de la figura 3.2.1.

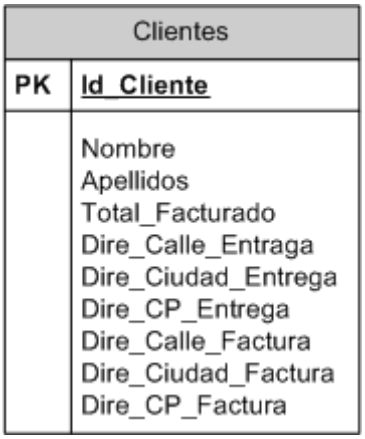


Figura 3.2.2: Agregación, tabla sencilla.

Como consideraciones a este mapeo, se tienen [10]:

- *Rendimiento*. Para obtener un rendimiento óptimo la solución debe permitir obtener un objeto en un único acceso a la base de datos evitando el uso de operaciones de tipo *join*. Los accesos a la base de datos deben utilizar un número mínimo de páginas para economizar el ancho de banda E/S.
- *Mantenibilidad*. En caso de que un tipo de objeto componente esté agregado en más de un tipo de objeto disminuye la facilidad de mantenimiento de la base de datos debido a que los cambios sobre el tipo de objeto se han de reflejar en el resto de tipos de objetos en los que ha sido agregado, con lo cual, se han de modificar todas las tablas que corresponden a esos tipos de objetos.
- *Consistencia de la base de datos*: La agregación implica que el ciclo de vida del objeto agregado está emparejado con el ciclo de vida del objeto componente. Esto debe estar garantizado bien por la base de datos o por la aplicación que la gestiona.

Como consecuencias de este mapeo se consideran:

- *Rendimiento*. Esta solución es óptima en términos de rendimiento dado que tan solo se ha de acceder a una tabla para obtener el objeto componente y todos sus objetos agregados. Pero por otro lado, los campos de los atributos de los objetos agregados tienden a incrementar el número de páginas recibidas con cada acceso, lo cual puede provocar un gasto de ancho de banda E/S elevado.
- *Mantenimiento y flexibilidad*. Si el objeto componente está agregado a más de un objeto, el diseño resultante resulta difícil de mantener dado que cada cambio del objeto componente implica la modificación de todos los objetos a los que ha sido agregado.
- *Consistencia de la base de datos*. Los objetos componentes son automáticamente borrados al borrarse los objetos compuestos. Para ello

no es necesario incluirlo en el código de la aplicación ni utilizar *triggers* de la base de datos.

- *Consultas ad-hoc*: El formular determinadas consultas puede resultar muy complicado. Por ejemplo, consulta que muestra de todos los objetos de Dirección.

Cuando se implementa el patrón, se ha de tener en cuenta:

- *Notación*. Conviene diferenciar los nombres de los atributos del objeto compuesto y de los objetos componentes.
- *Tamaño físico de la página de la base de datos*. El efecto positivo de agregar objetos en la misma tabla puede ser perjudicado si los atributos del objeto componente comienzan una nueva página de la base de datos. En esta situación se requiere la lectura de dos páginas de la base de datos en lugar de una.

b) *Mapeo por clave extranjera*

Consiste en usar una tabla para el objeto componente. Para ello se utiliza un *Identificador Sintético de Objeto* ISO en la tabla del objeto compuesto (agregado) que hace las veces de clave extranjera. En nuestro caso asociamos el ISO a un valor no repetible de la tabla. El objeto compuesto se mapea a una tabla. El objeto componente a otra que contiene el ISO. Este ISO es referenciado por la clave principal (ISO_Objeto_compuesto) en su tabla. En la figura 3.2.3 se puede observar como resulta el mapeo por clave extranjera de la agregación mostrada en la figura 3.2.1.

Recuperar un objeto de un cliente cuesta ahora tres operaciones de acceso a la base de datos (una para Cliente, una para cada Tipo_Direccion, Dire_Entrega y Dire_Factura), frente a un acceso para el patrón de tabla sencilla, además de contar con una relación entre ambas tablas.

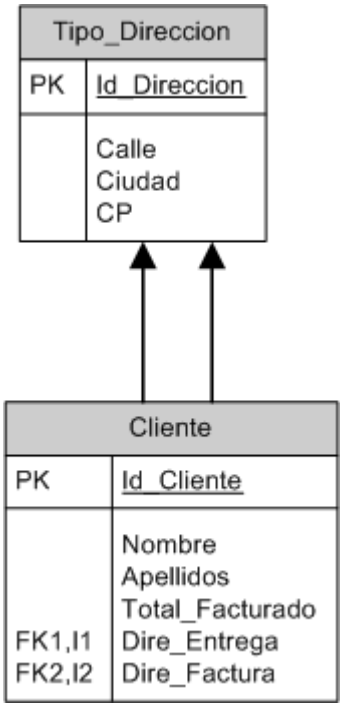


Figura 3.2.3: Agregación, clave extranjera.

Como consecuencias de este mapeo, se obtienen:

- *Rendimiento*. Se ha de tener en cuenta que si se desea obtener el objeto agregado es necesario realizar al menos dos accesos a la base de datos y una operación *join*. Esto significa una pérdida de rendimiento respecto al mapeo en tabla única; por el contrario, si rara vez se accede al objeto agregado la solución de mapeo utilizando clave extranjera ofrece un rendimiento similar al de mapeo en tabla única.
- *Mantenimiento.* El almacenar los objetos componentes en tablas diferentes al objeto compuesto facilita el mantenimiento y aumenta la flexibilidad del mapeo.
- *Consistencia de la base de datos.* Los objetos componentes no son automáticamente borrados cuando se borran los objetos compuestos a los que han sido agregados. Para llevar a cabo esta tarea es necesario que la aplicación esté preparada para realizar estas tareas de mantenimiento o se utilicen *triggers* de la base de datos. Esto supone también un problema de implementación. Se ha de seleccionar una de estas dos opciones.
- *Consultas ad-hoc.* Factorizando los objetos agregados en tablas diferentes resulta sencillo formular consultas.

3.3 Asociación

Para el mapeo de asociación entre clases se utilizan dos patrones: asociación con clave extranjera y tabla de asociación. En la figura 3.3.1 se muestra el diagrama de clases de asociación que se desea mapear en tablas.

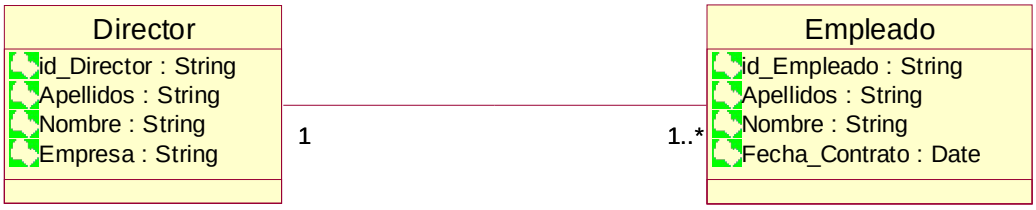


Figura 3.3.1: Asociación.

a) Asociación con clave extranjera

Este patrón permite mapear asociaciones 1: n en tablas relacionales. Esto se realiza insertando un campo en la tabla donde se almacena el objeto dependiente que contiene el identificador del objeto propietario. Nótese que en la figura no aparece una navegabilidad de la relación entre una clase y otra. El objeto dependiente es aquel que en la relación tiene la multiplicidad n, o que según el patrón recibe la flecha de navegabilidad. Este identificador puede ser representado por una clave de la base de datos o por un ISO. En la figura 3.3.2 se puede observar el resultado de mapear en tablas la asociación mostrada en la figura 3.3.1.

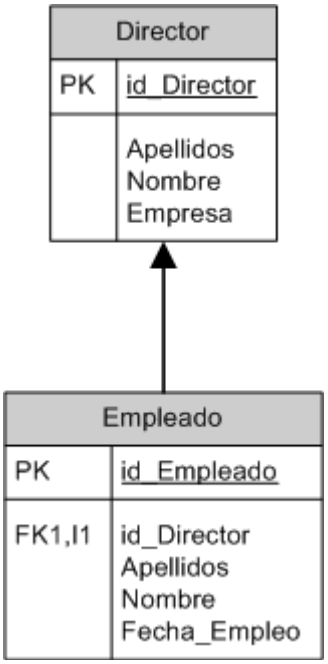


Figura 3.3.2: Asociación, clave extranjera.

Como consecuencias de este mapeo, se obtienen:

- *Rendimiento de lectura.* Para la lectura de un objeto con sus objetos dependientes se ha de utilizar una consulta con el operador *join.*
- *Rendimiento de escritura.* Se necesitan tantas operaciones de escritura como objetos dependientes han cambiado.
- *Rendimiento y redundancia contra mantenimiento y formas normales.* El esquema de este mapeo es un esquema muy común en las bases de datos relacionales, de modo que no colisiona con las formas normales obteniendo así un coste de mantenimiento razonable.
- *Consumo de espacio.* Es prácticamente el óptimo salvo por el campo de clave extranjera que hay que insertar en las tablas dependientes.
- *Consultas ad-hoc.* Dado que su esquema de mapeo es muy común en las bases de datos relacionales, las consultas *ad-hoc* no serán más fáciles ni más difíciles que las de cualquier aplicación en bases de datos relacionales.
- *Estilo de la Aplicación.* Es el mapeo que mejor se adapta a aplicaciones de tipo relacional. No se deben utilizar bases de datos relacionales para uso como CAD. Resolver una asociación cuesta una operación de tipo *join* o un segundo acceso a la base de datos.
- *Integración en sistemas de herencia*. La mayoría de los sistemas de herencia usa exactamente este tipo de mapeo, convirtiendo asociaciones 1: n en objetos sin ser fuente de nuevos problemas.

b*) Tabla de asociación*

Este patrón se emplea para mapear asociaciones n: m en bases de datos relacionales. Para ello se crea una tabla que contiene dos campos que hacen referencia a los ISO de los dos objetos que participan en la asociación. Estos dos objetos se han mapeado en otras dos tablas. En la figura 3.3.3 se puede observar el resultado de aplicar este patrón de mapeo a la asociación mostrada en la figura 3.3.1.

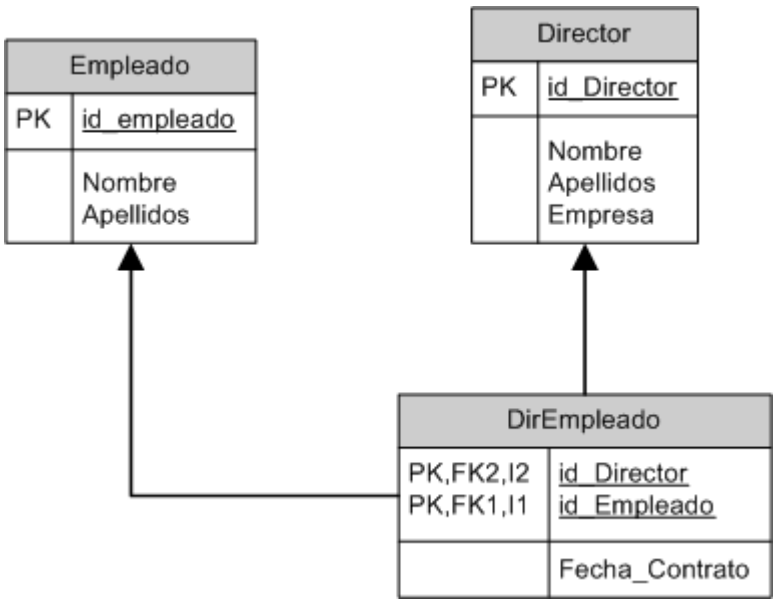


Figura 3.3.3: Asociación, tabla de asociación.

Las consecuencias de este tipo de mapeo son las mismas que las del mapeo por clave extranjera.

Cuando se implementa el patrón, se ha de tener en cuenta dos aspectos:

- *Precarga de objetos.* Para la mayoría de los casos, si se conoce por adelantado la necesidad de acceder a todos los objetos dependientes se pueden obtener todos los datos empleando una operación *join* y construir el objeto propietario y los objetos dependientes del resultado de una consulta.

3.4 Herencia

Existen varias formas de realizar el mapeo de herencia en una base de datos relacionales y se van a estudiar tres formas de hacerlo. En la práctica se suele recurrir al empleo de formas mixtas. La figura 3.4.1 muestra el diagrama de clases de herencia que se desea mapear a tablas relacionales.

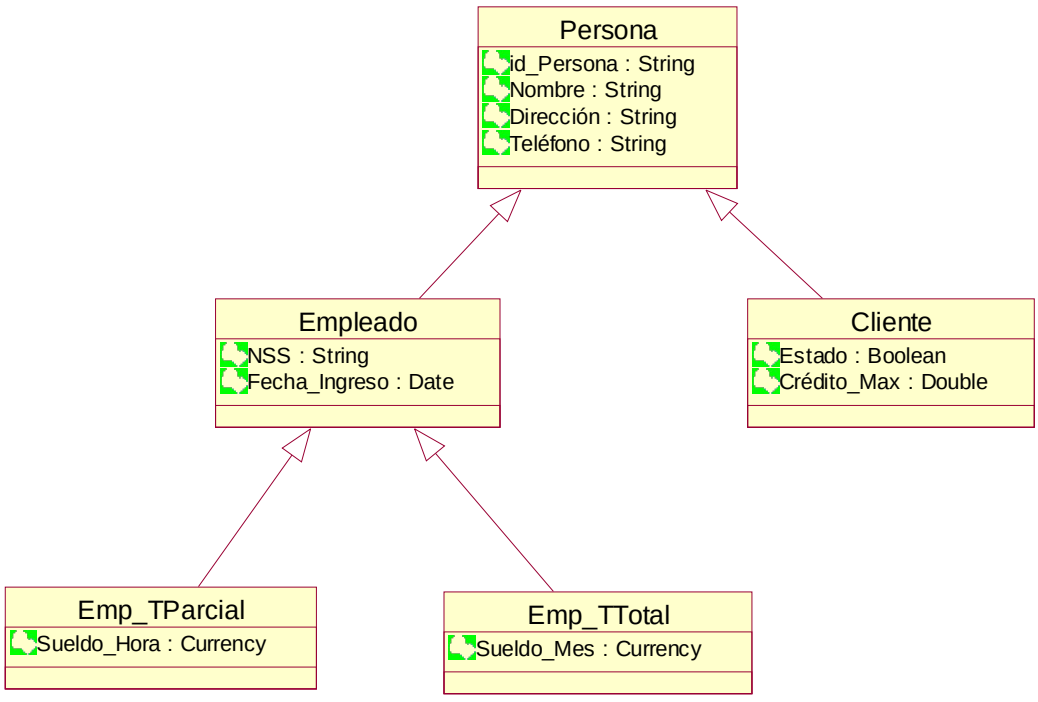


Figura 3.4.1: Herencia.

a) *Herencia de árbol – una tabla*

Consiste en agrupar los atributos de todos los objetos pertenecientes a la jerarquía de herencia en una única tabla, utilizando *NULL* como valor para los campos de los registros que no hagan uso de ellos. La figura 3.4.2 muestra cómo queda la tabla para este tipo de mapeo.

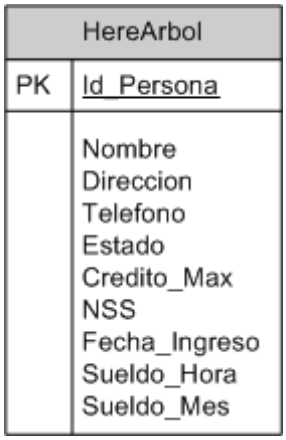


Figura 3.4.2: Herencia , herencia de árbol - una tabla.

Al igual que en los casos anteriores hemos de tener en cuenta parámetros como el rendimiento, velocidad de lectura, escritura, facilidad de mantenimiento, flexibilidad…, pero dada la naturaleza de la herencia se han de añadir otra serie de parámetros a tener en cuenta como son:

- *Lectura polimórfica y espacio consumido contra rendimiento en escritura y actualización.* En una jerarquía de herencia es necesario soportar consultas donde todos los objetos cumplan algún criterio. El resultado de esta consulta es polimórfico. Las soluciones que dan un mejor soporte a las consultas polimórficas son aquellas que más espacio ocupan y suponen un menor rendimiento de escritura.
- *Esquemas de bloqueo de la base de datos.* Algunas bases de datos implementan bloqueos a nivel de página solamente o pueden ser programadas para escalar tipos de bloqueo fácilmente. En ese caso hay que andar con cuidado para que el tráfico en una tabla simple no exceda un límite impuesto por excesivos bloqueos y un rendimiento bajo.
- *Profundidad de la jerarquía de herencia.* Algunas de las soluciones que se pueden adoptar pueden trabajar muy bien en jerarquías de herencia planas y en cambio funcionar muy mal en jerarquías de herencia profundas.
- *Facilidad de mantenimiento.* Aquellos mapeos que dividen los atributos de los objetos en varias tablas pueden ser los que ofrezcan un mejor rendimiento en lecturas polimórficas, pero por el contrario resultan muy difíciles de mantener en caso de que se añadan o eliminen atributos de los objetos. La evolución del esquema necesita tener en cuenta qué datos son replicados a través del modelo físico. Otro caso de mantenimiento puede ser la inserción o borrado de clases dentro de la jerarquía de herencia.

- *Consultas definidas por el usuario*. Si se desea ofrecer al usuario la oportunidad de formular sus propias consultas, es necesario asumir que los mapeos de las tablas son incomprensibles desde la perspectiva del usuario.

Como consecuencias de este mapeo, se obtienen las siguientes:

- *Rendimiento de escritura y actualización*. Dado que todos los atributos del objeto se encuentran almacenados en una tabla, es posible leer y escribir sobre los objetos que heredan en un único acceso a la base de datos.
- *Rendimiento de lectura polimórfica*. Dado que todos los objetos que pertenecen a una misma jerarquía de herencia comparten la misma tabla, la lectura polimórfica es directa.
- *Espacio consumido*. Como se puede observar en la figura x.x.6, almacenar los atributos de los objetos requiere más espacio del absolutamente necesario. Este gasto de espacio depende de la profundidad de la herencia. Cuanto mayor sea la profundidad de la herencia y mayor sea la diferencia entre la unión de todos los atributos y los atributos del objeto medio (es padre y a la vez hijo), mayor será el consumo de espacio.
- *Balanceo de carga de la base de datos a través de las tablas*. Mapear demasiadas tablas en un única tabla puede ser causa de un mal rendimiento. Los orígenes de este problema se pueden encontrar en el comportamiento de la base de datos:
    - Si la base de datos utiliza bloqueo a nivel de página, demasiado tráfico en una sola tabla puede ralentizar el acceso. Si se produce mucho tráfico sobre una tabla es de esperar una degradación del rendimiento e incluso interbloqueos.
    - Demasiados bloqueos en una tabla puede acabar en una escalada de bloqueos.
    - Algunas clases necesitan un índice secundario para acelerar su búsqueda. Si se implementan varias clases en una sola tabla se deberán crear índices en esa tabla. Demasiados índices en una sola tabla pueden provocar que las actualizaciones sean muy lentas dado que todos los índices deberán ser actualizados.
- *Coste de mantenimiento*. El mapeo es directo y sencillo, el esquema de evolución también es directo y sencillo mientras la jerarquía de la herencia no sea muy profunda.
- *Consultas ad-hoc*. Las consultas son muy sencillas de construir.

Cuando se implementa el patrón, se ha de tener en cuenta los siguientes aspectos.

- *Consideración de mapear todos los objetos en una tabla*. Este mapeo se puede utilizar para almacenar todos los tipos de objetos en una tabla, lo que provoca un elevado tráfico sobre esa tabla. Para pequeñas aplicaciones resulta un mapeo factible y muy flexible.
- *Consumo de espacio*. Caso de que la base de datos admita el empaquetamiento de los valores *NULL*, este mapeo resulta más atractivo ya que evita el gasto innecesario de espacio en el almacenamiento de valores *NULL*.

b) *Mapeo de una clase, una tabla*

Se mapean los atributos de cada clase en tablas separadas. Para ello se inserta un *Identificador Sintético de Objeto* en cada tabla para enlazar las filas de las clases derivadas con sus correspondientes filas en las tablas padre. En la figura 3.4.3 se observa cómo resulta de aplicar este patrón de mapeo al ejemplo descrito en la figura 3.4.1.

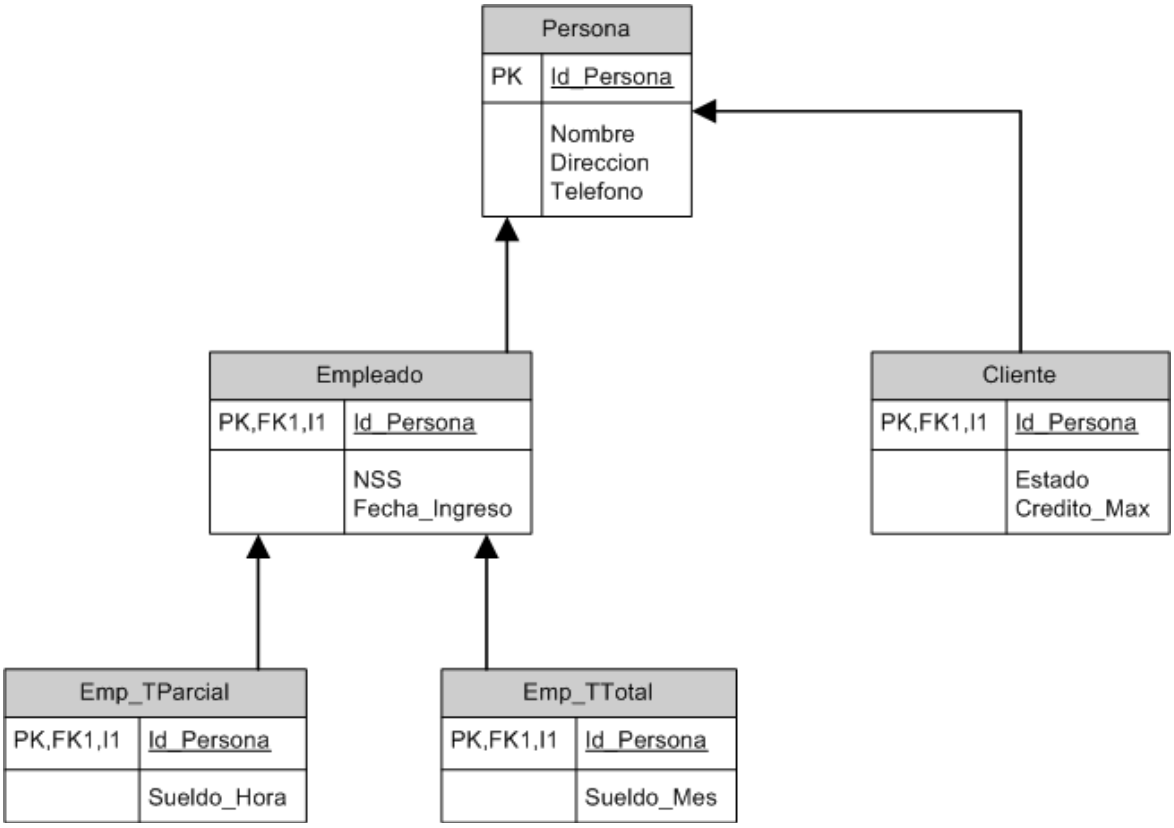


Figura 3.4.3: Herencia, mapeo de una clase, una tabla.

Como consecuencias de este mapeo, se obtienen:

- *Rendimiento de escritura y actualización*. Este patrón proporciona un mapeo muy flexible, pero no es el que proporciona un mejor rendimiento. Como se puede observar en la figura 3.4.3la lectura de un objeto *Emp_TParcial* requiere tres operaciones de lectura: una para la tabla *Emp_TParcial*, otra para la tabla Empleado y una más para la tabla Persona. La escritura requiere tres operaciones de escritura en la base de datos, cada una de ellas actualizando uno o más índices. Este mapeo es costoso en términos de operaciones en la base de datos en tareas intensivas de escritura y actualización. Este coste se ve incrementado con el aumento de la profundidad de la jerarquía de la herencia.
- *Rendimiento de lectura polimórfica*. En este tipo de mapeo la lectura polimórfica se realiza accediendo a una sola tabla.
- *Espacio consumido*. El espacio que consume es prácticamente el estrictamente necesario salvo por la necesidad de añadir los ISO.

- *Consultas ad-hoc*. Dado que generalmente se requiere realizar accesos a más de una tabla para recibir información acerca de los objetos, las consultas a realizar pueden resultar complicadas para usuarios inexpertos.
- *Carga en la base de datos por las tablas raíz.* Debido al patrón seguido para realizar el mapeo, las tablas raíz de los objetos sufren una elevada carga, esto puede generar cuellos de botella en la base de datos.

c) Una ruta de herencia, una tabla

Consiste en mapear cada clase en una tabla, de modo que para el ejemplo descrito en la figura 3.4.1 se crean tres tablas independientes. En la figura 3.4.4 se puede observar el resultado de ese mapeo.

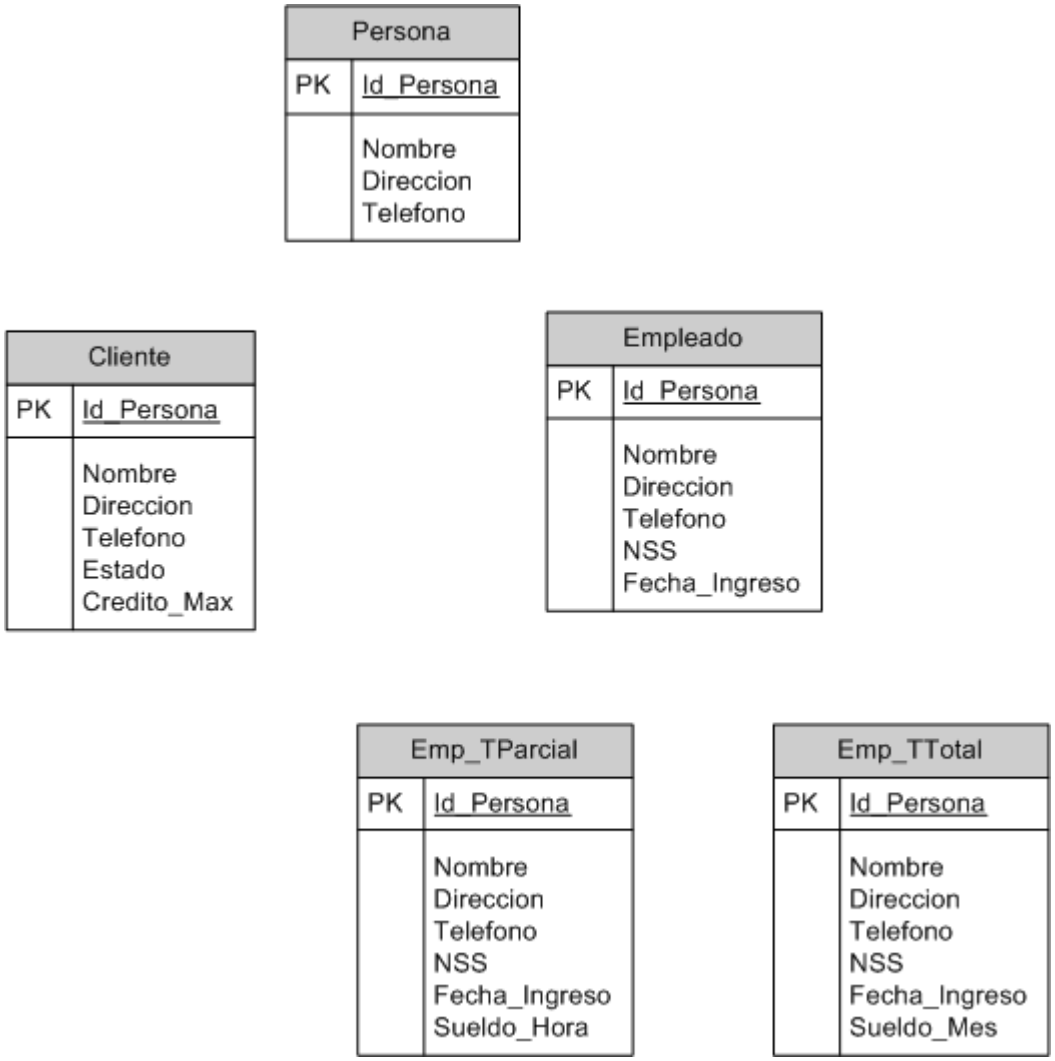


Figura 3.4.4: Herencia, una ruta de herencia, una tabla.

Como consecuencias de este mapeo, se obtienen:

- *Rendimiento de escritura y actualización.* En este patrón de mapeo para realizar una operación de lectura o escritura sobre un objeto se requiere un solo acceso a la base de datos.
- *Lectura polimórfica.* Una lectura polimórfica de los objetos requiere el acceso a varias tablas. Lo cual resulta más costoso comparando con los otros mapeos.
- *Espacio consumido.* Con este tipo de mapeo se consigue alcanzar un consumo de espacio óptimo al no haber datos redundantes.
- *Coste de mantenimiento.* Insertar una nueva subclase significa tener que actualizar todas las consultas de búsqueda polimórfica, de modo que la estructura de las tablas se vuelve imposible de mantener. Añadir o eliminar atributos de una superclase implica realizar cambios en todas las clases derivadas de esta. Esto puede modificar las consultas de búsqueda polimórfica, si éstas son estáticas en lugar de ser dinámicamente generadas desde un diccionario. Por lo tanto, este patrón necesita el soporte de generadores y consultas dinámicas para poder ser sostenible.
- *Consultas ad-hoc.* Dado que el mapeo generalmente requiere acceder a más de una tabla para realizar consultas de búsqueda polimórfica, las consultas *ad-hoc* para búsqueda polimórfica resultan complejas de escribir para usuarios inexpertos. Las consultas en clases individuales son triviales.
- *Carga en las tablas raíces.* No existen cuellos de botella en las tablas cercanas a la raíz de la jerarquía de la herencia. Acceder a un objeto bloquea únicamente una tabla.

4. Análisis y Diseño

4.1 Requisitos

Los requisitos se van a dividir en dos tipos:

- Funcionales: los requeridos por el usuario.
- No funcionales: los requeridos para que la herramienta funcione.

4.1.1 Requisitos funcionales

- Selección de análisis manual.
  - Creación de tablas.
  - Visión detallada de las tablas en la base de datos.
  - Creación de consultas personalizadas y caso de incluir datos concretos configuración del número de apariciones.
- Selección de análisis automático.
  - Visualización gráfica y selección de los diferentes patrones de mapeo.
  - Selección de las diferentes consultas a analizar.
- Configuración de las pruebas a realizar:
  - Número de registros sobre los que trabajar.
  - Visualización o no de los resultados de las consultas.
  - Número de repeticiones por consulta.
- Selección del SGBDR sobre el que realizar las pruebas.
- Configuración de los datos de acceso al SGBDR.
- Inicio y visualización en tiempo real de las pruebas de rendimiento.
- Visualización y tratamiento de los datos obtenidos:
  - Generación de gráficas y tablas.
    - Media.
    - Media armónica.
    - Moda.
    - Desviación estándar.
  - Posibilidad de almacenar en formato físico los resultados.
  - Cargar los datos obtenidos de pruebas anteriores.
  - Comparar los resultados almacenados en diferentes pruebas.
  - Exportar los datos.

4.1.2 Requisitos no funcionales

- Al tratarse de una herramienta con orientación didáctica el interfaz ha de ser sencillo e intuitivo.
- SGBDR en funcionamiento y accesibles desde el ordenador:
  - Oracle.
  - Postgresql

  - MySQL.
- Usuarios con los permisos adecuados en los diferentes SGBDR:
  - Creación y borrado de tablas.
  - Consultas SELECT, INSERT, DELETE y UPDATE.
- Conector ODBC configurado para el acceso a Access.
- Conectores JDBC para Oracle, Postgresql y MySQL.
- Ficheros de texto con las consultas de creación de tablas para los distintos SGBDR.
- Ficheros de texto con las consultas predeterminadas para la medición de rendimiento.
- Sistema operativo que cuente con una máquina virtual de Java.

4.2 Casos de uso

Analizando los requisitos de la herramienta se determina que únicamente es necesario un único actor al que llamaremos usuario y tres casos de uso generales, los cuales se pueden ir desglosando en diferentes casos de uso detallados.

4.2.1 Casos de uso generales

A continuación se detallan los tres casos de uso generales.

- Caso de uso configuración:
  - Descripción: El usuario configura los diferentes parámetros de las pruebas de rendimiento a realizar.
  - Precondición: se ha presionado el botón de inicio de la herramienta.
  - Flujo:
    - Se selecciona el SGBDR que se desea emplear.
    - Se configura los datos de conexión al SGBDR.
    - Selección de número de registros y de repeticiones.
    - Selección del tipo de prueba.
  - Postcondición: la herramienta está preparada para realizar las mediciones de rendimiento.
- Caso de uso realización de las pruebas de rendimiento:
  - Descripción: el usuario inicia la ejecución de las pruebas de rendimiento.
  - Precondición: se ha debido configurar las pruebas de rendimiento a realizar.
  - Flujo:
    - Pulsa el botón “Start”.
    - Visualización en tiempo real de las consultas y los tiempos requeridos.
  - Postcondición: se dispone en memoria de todas operaciones realizadas así como de los resultados obtenidos.
- Caso de uso tratamiento de resultados:
  - Descripción: el usuario puede visualizar, almacenar, cargar y comparar los resultados obtenidos en diferentes pruebas de rendimiento.
  - Precondición: se ha de contar con resultados de pruebas, ya sea en memoria volátil tras una prueba de rendimiento o almacenado en memoria física.

- Flujo:
  - Normal:
    - Pulsar el botón de análisis de la herramienta.
    - Carga de resultados almacenados.
    - Comparación entre diferentes colecciones de datos.
  - Alternativo:
    - Visualización de los resultados obtenidos tras las pruebas de rendimiento.
    - Carga de resultados almacenados.
    - Comparación entre diferentes colecciones de datos.
- Postcondición: el usuario dispone de manera legible los resultados obtenidos en las diferentes pruebas de rendimiento.

4.2.2 Casos de uso detallados

Los tres casos de uso generales los podemos dividir en varios casos de uso más detallados.

- Configuración:
  - Caso de uso configuración del conector a la base de datos.
    - Descripción: configurar los parámetros de acceso a los diferentes SGDBRs
    - Precondición: acceder al área de configuración “Data Base” de la herramienta.
    - Flujo:
      - Seleccionar el SGBDR.
      - Pulsar botón de configuración del conector.
      - Introducir los datos de configuración.
      - Comprobar el correcto funcionamiento de la conexión.
      - Seleccionar la aplicación de la emulación de borrado y actualización en cascada.
    - Postcondición: ya es posible acceder al SGBDR objeto de las pruebas.
  - Caso de uso configuración básica de las pruebas.
    - Descripción: Configurar el número de registros sobre los que se van a realizar las pruebas y el número de repeticiones de cada consulta.
    - Precondición: acceder al área de configuración “Settings”de la herramienta.
    - Flujo:
      - Indicar el número de registros sobre el que se va a trabajar.
      - Indicar el número de repeticiones de cada consulta.
    - Postcondición: un aspecto de la configuración ya no está en su valor por defecto sino que se encuentra conforme los deseos del usuario.
  - Caso de uso configuración general.
    - Descripción: selección del tipo de pruebas a realizar, manual o automática.

        - Precondición: acceder al área de configuración "General" de la herramienta.
        - Flujo:
            - Seleccionar el tipo de prueba.
            - Configuración particular de cada tipo de prueba.
        - Postcondición: aspectos generales del tipo de pruebas a realizar ya se encuentran configurados.
- Realización de las pruebas de rendimiento:
    - Caso de uso iniciación de las pruebas de rendimiento:
        - Descripción: visualización y control de las consultas realizadas.
        - Precondición: las pruebas han sido configuradas.
        - Flujo:
            - Inicio de las pruebas.
        - Postcondición: las pruebas de rendimiento se están ejecutando.
    - Caso de uso pausa:
        - Descripción: se realiza una parada temporal de la prueba.
        - Precondición: las pruebas de rendimiento están siendo realizadas.
        - Flujo:
            - Pulsar el botón de parada.
        - Postcondición: las pruebas de rendimiento se encuentran momentáneamente paradas.
    - Caso de uso continuar:
        - Descripción: se continúa con la ejecución de las pruebas de rendimiento.
        - Precondición: las pruebas de rendimientos se encuentran paradas.
        - Flujo:
            - Pulsar botón de continuar.
        - Postcondición: las pruebas de rendimiento se están ejecutando.
    - Caso de uso parar:
        - Descripción: interrumpimos definitivamente las pruebas de rendimiento.
        - Precondición: las pruebas de rendimiento están siendo realizadas.
        - Flujo:
            - Pulsar el botón de parada.
        - Postcondición: la ejecución de las pruebas de rendimiento se encuentran detenidas definitivamente.
    - Caso de uso gestión de resultados:
        - Descripción: una vez finalizadas las pruebas de rendimiento podemos decidir qué hacer con los datos obtenidos.
        - Precondición: se ha terminado de realizar las pruebas de rendimiento.
        - Opciones:
            - Copiar al portapapeles el registro de las pruebas realizadas.
            - Salvar los datos.
            - Visualizar resultados
            - Salir.
- Tratamiento de resultados:

- Caso de uso configuración visualización:
  - Descripción: configurar el dato a visualizar, la escala de los registros y el tipo de visualización.
  - Precondición: contar con los resultados de las pruebas de rendimiento.
  - Opciones
    - Media, media harmónica, moda, desviación estándar.
    - Gráfica o tabla.
    - Visualización lineal o logarítmica.
    - Dibujar aproximación.
  - Postcondición: el usuario ha analizado los resultados obtenidos.

4.3 Clases

Gracias a los casos de uso definidos en la sección anterior vamos poder diseñar e una manera eficiente y sencilla el conjunto de paquetes y de clases que formaran nuestra herramienta.

Dividimos la aplicación en tres paquetes que se corresponden con los casos de uso generales y un cuarto paquete que nos permitirá realizar el análisis detallado de las tablas contenidas en la base de datos. Podemos ver su representación en la figura 4.3.1

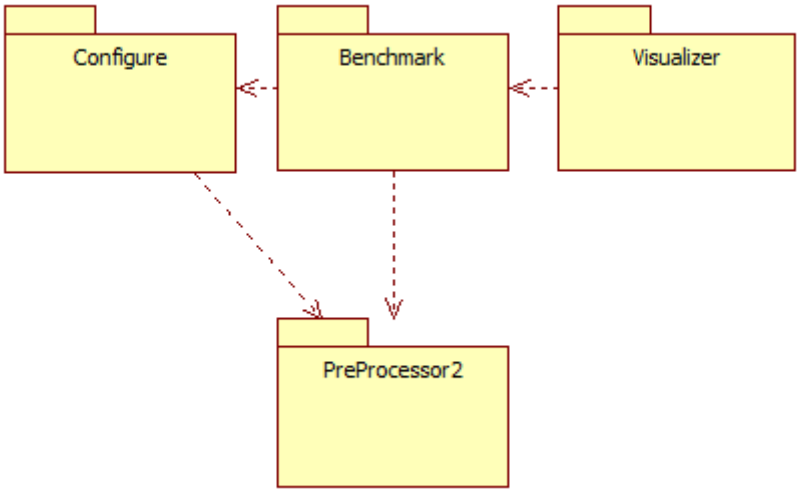


Figura 4.3.1: Conjunto de paquetes de la herramienta.

Una vez se ha definido los paquetes procedemos a detallar las clases que se incluirán dentro de cada paquete necesarias para satisfacer los requisitos y los casos de uso previamente definidos.

Paquete PreProcessor2

Se trata de un paquete cuya función va a ser la de analizar detalladamente las tablas de una base de datos y proveer de esa información al resto de clases que lo necesiten. Su nombre viene heredado del preprocesador con el que ya venía equipada la herramienta en su versión inicial, pero dado que este no cumplía con los requisitos de flexibilidad y capacidad de conectarse a varios SGBDR distintos de MS Access se opta por programar un preprocesador totalmente nuevo.

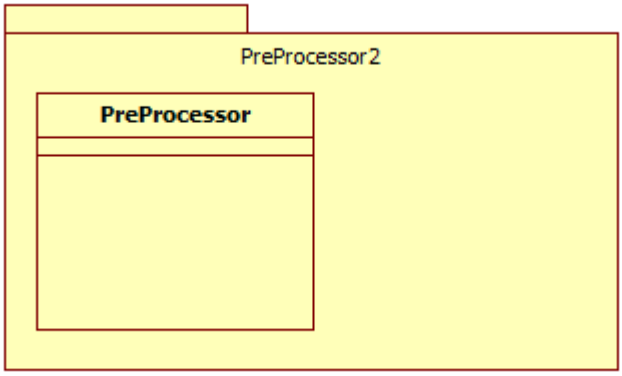


Figura 4.3.2: Paquete PreProcessor2.

En la siguiente figura podemos visualizar el diseño de la clase.

Figura 4.3.3: Clase PreProcessor2.

La clase únicamente contendrá un método que tendrá como argumento las consultas de creación de tablas y los datos relativos a la conexión con la base de datos, como salida proporcionará todos los datos necesarios para describir en detalle todas las tablas de la base de datos, campos, tipos, relaciones, índices, claves..., esta información será aprovechados por las clases de la herramienta que lo necesiten.

Paquete Configure

En este paquete se van a encontrar todas las clases que nos permitirán realizar la configuración de las pruebas de rendimiento a realizar. Se tratará básicamente de un conjunto de ventanas y paneles que configurarán el valor de una serie de parámetros que más tarde serán utilizados por el paquete Benchmark.

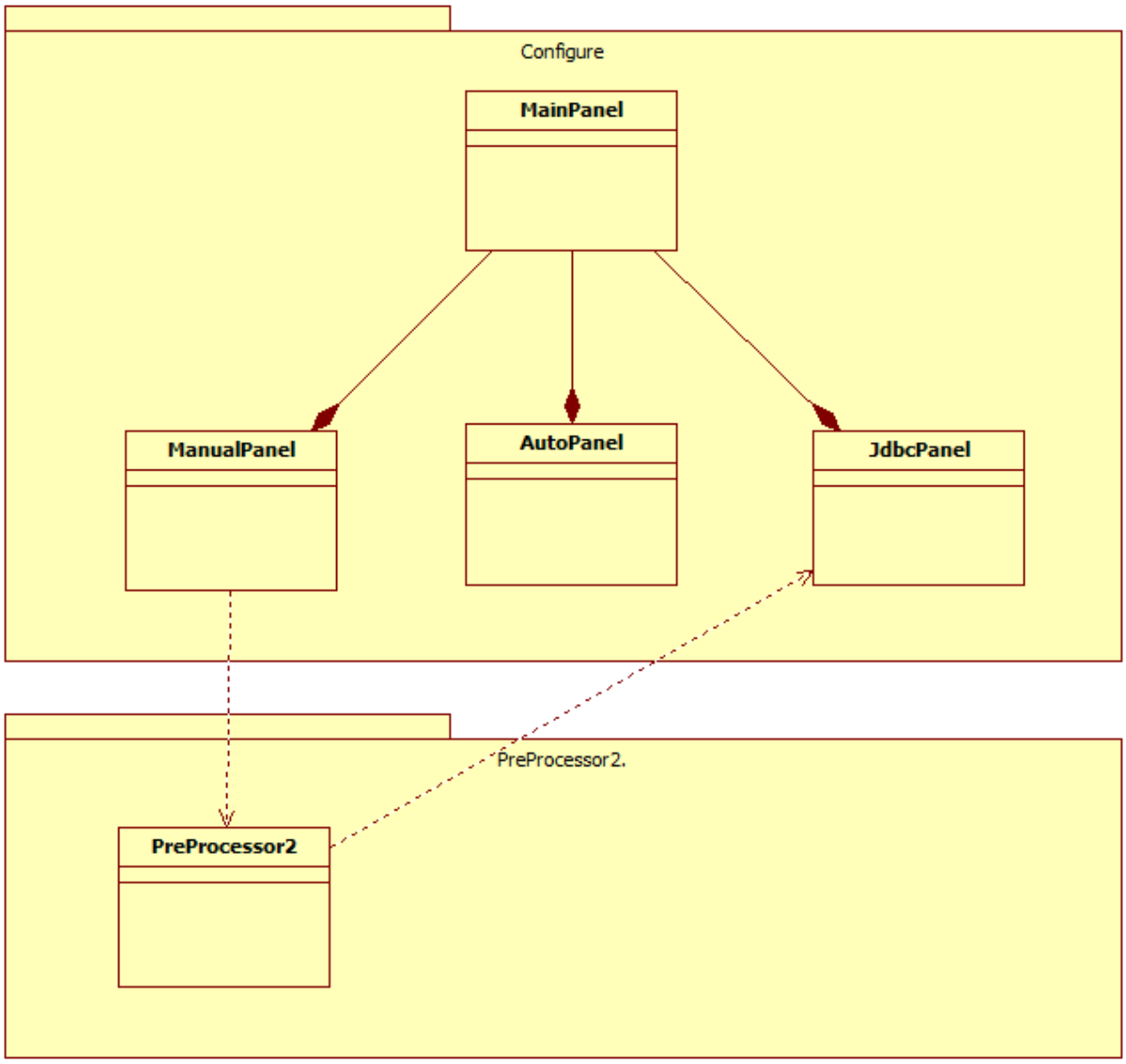


Figura 4.3.4: Paquete Configure.

En la figura4.3.4 podemos visualizar las clases que forman el paquete Configure, una de ellas será el panel principal (MainPanel) donde se encontrarán todos los elementos comunes a todos los tipos de pruebas salvo la configuración de los conectores jdbc, que aun tratándose de un elemento común se decide separarlo en su propia clase para de este modo añadir modularidad y en caso de querer añadir un nuevo conector sólo se tenga que acudir a esa clase. Otra clase se corresponderá con la configuración de las pruebas manuales (ManualPanel), otra clase para la configuración de las pruebas automatizadas (AutoPanel) y por último la clase JdbcPanel donde como ya se ha comentado anteriormente nos permite configurar los conectores jdbc.

- MainPanel: Creación del panel y un método que nos permita visualizarlo en pantalla. El panel principal contendrá al resto de paneles.

Figura 4.3.5: Clase MainPanel.

- AutoPanel: Panel con los elementos de configuración referentes a las pruebas automatizadas.
    - SettingFrameAuto: creación del panel con todos los elementos necesarios para la configuración y visualización de las diferentes pruebas automatizadas. Recibe como parámetro de entrada la base de datos sobre la que va a actuar.
    - actionPerformed: gestiona todos los eventos sobre los diferentes elementos contenidos en el panel.
    - lecturaFicheroCreates: nos permite leer los ficheros de texto que contienen las consultas de creación de tablas. Como argumentos tendrá información acerca de la base de datos y el nombre del fichero en el que se encuentran las sentencias SQL de creación de tablas. Como salida nos proporciona el texto de las consultas.
    - lecturaFicheroQueries: Como argumentos tendrá información acerca de la base de datos y el nombre del fichero en el que se encuentran las consultas objeto de medición. Como salida nos proporciona el texto de las consultas.
    - lecturaRegistrosEspeciales: nos permite leer datos de configuración de las pruebas de rendimiento como puede ser modo y número de registros con valores concretos que se van a emplear en el proceso de medición.

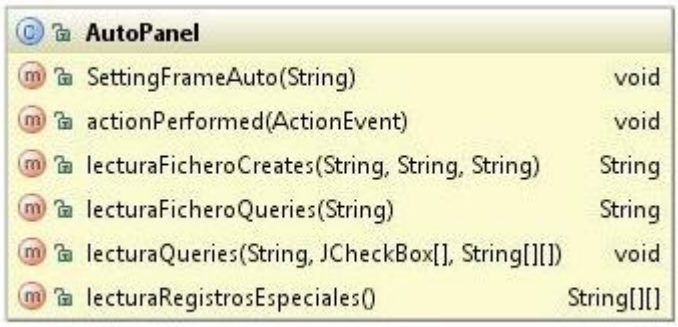


Figura 4.3.6: Clase AutoPanel.

- ManualPanel: Panel con los elementos de configuración referentes a las pruebas manuales, presenta una dependencia con la clase Preprocessor2 debido a que es necesario mostrar al usuario la estructura de las tablas de la base de datos para así poder construir adecuadamente las consultas.
    - actionPerformed: gestiona todos los eventos sobre los diferentes elementos contenidos en el panel.
    - actualizarTabla: actualizar los datos referentes a las tablas en tiempo real en caso de que el usuario modifique las consultas de creación de tablas. Nos

devuelve un entero que indica el resultado de la actualización y así poder gestionar posibles errores de sintaxis en las consultas de creación.
  - propertyChange: método para gestionar cambios de elementos que interaccionan entre sí.
  - stateChange: método para gestionar cambios de elementos que interaccionan entre sí.
  - Visualizar: método para la gestión de la visualización, como entrada recibe información referente a la base de datos.

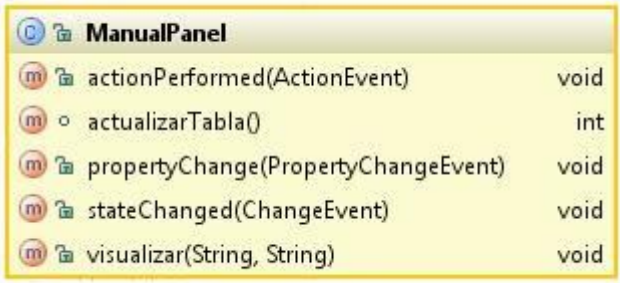


Figura 4.3.7: Clase ManualPanel.

- JbdcPanel: panel con todos los elementos referentes a la configuración de los distintos conectores a SGBDs.
  - JdbcPanel: método que gestiona la información de todos los elementos del panel relativo a la configuración de los conectores a los SGBD.

Figura 4.3.8: Clase JdbcPanel.

Paquete Benchmark

Este paqueteserá el responsable de realizar todas las tareas que sean necesarias para realizar las pruebas de rendimiento según se han configurado previamente en el paquete Configure. Una vez se realizan las pruebas, este paquete ofrece como salida los datos que utilizará el paquete Visualizer.

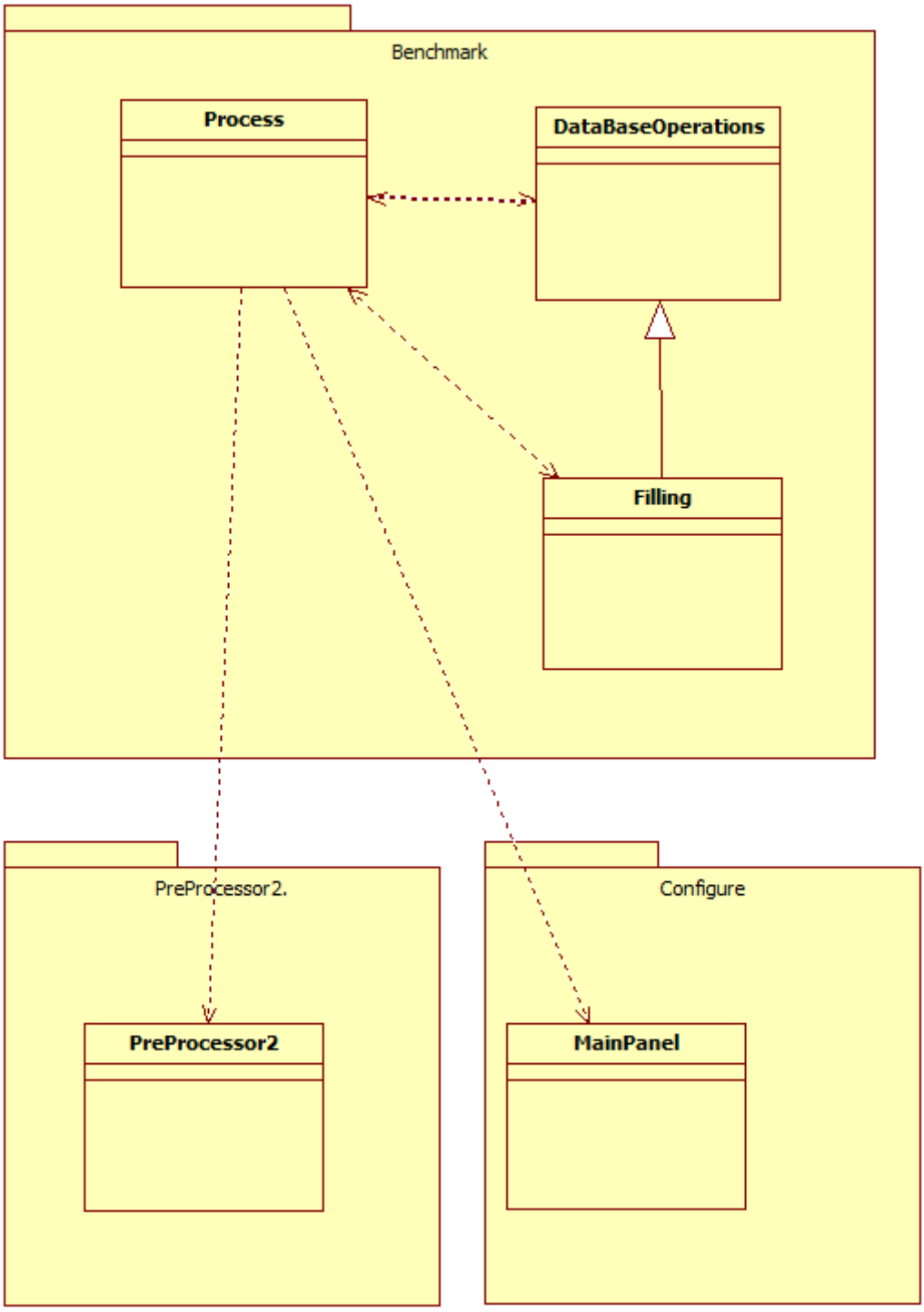


Figura 4.3.9: Paquete BenchMark

En la figura 4.3.9 podemos observar como el paquete se va a constituir por tres clases relacionadas entre sí, y una de ellas se relaciona con otra clase de otro paquete.

- DataBaseOperations: una clase que contendrá una serie de métodos que nos permitirán realizar una serie de operaciones básicas sobre una base de datos.

- EraseTable: nos permite borrar una tabla, como argumentos recibe los datos de la conexión y de la tabla a borrar.
- connectB: realiza la conexión a una base de datos. Como entrada tendrá información sobre la base de datos a la que va a conectarse.
- countRegisters: cuenta el número de registros de una tabla. Recibe información sobre la base de datos y la tabla, nos devolverá el número de registros.
- eraseAllTables: borra todas las tablas de la base de datos que se le especifique. Nos retorna códigos de error o éxito.

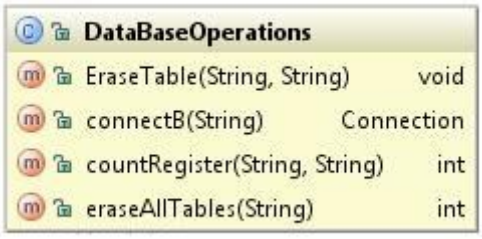


Figura 4.3.10: Clase DataBaseOperations.

- Filling: clase que se encargará de llenar de datos aleatorios todas las tablas de la base de datos. Esta clase hereda de dataBaseOperations debido a que va realizar muchas de las operaciones ya implementadas en dataBaseOperations. En la figura 4.3.11 podemos ver los métodos que implementa.

  - ComprobarHaIdoBien: nos permite comprobar si el proceso ha finalizado correctamente. Nos devuelve un mensaje en código.
  - PasarTexto: método de comunicación para comunicarse con la clase Process.
  - detectarTipoHerencia: método que ayuda a la detección del tipo de patrón de mapeo de herencia.
  - formatoTiempo: nos transforma medidas de tiempo a formatos de texto que se adecuan a la herramienta.

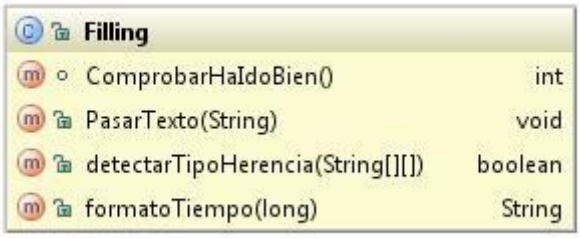


Figura 4.3.11: Clase Filling.

La gran complejidad de esta clase se encuentra en el propio constructor que para poder insertar los registros de una manera adecuada en tablas con configuraciones variadas ha de seguir el siguiente diagrama de flujo 4.3.12

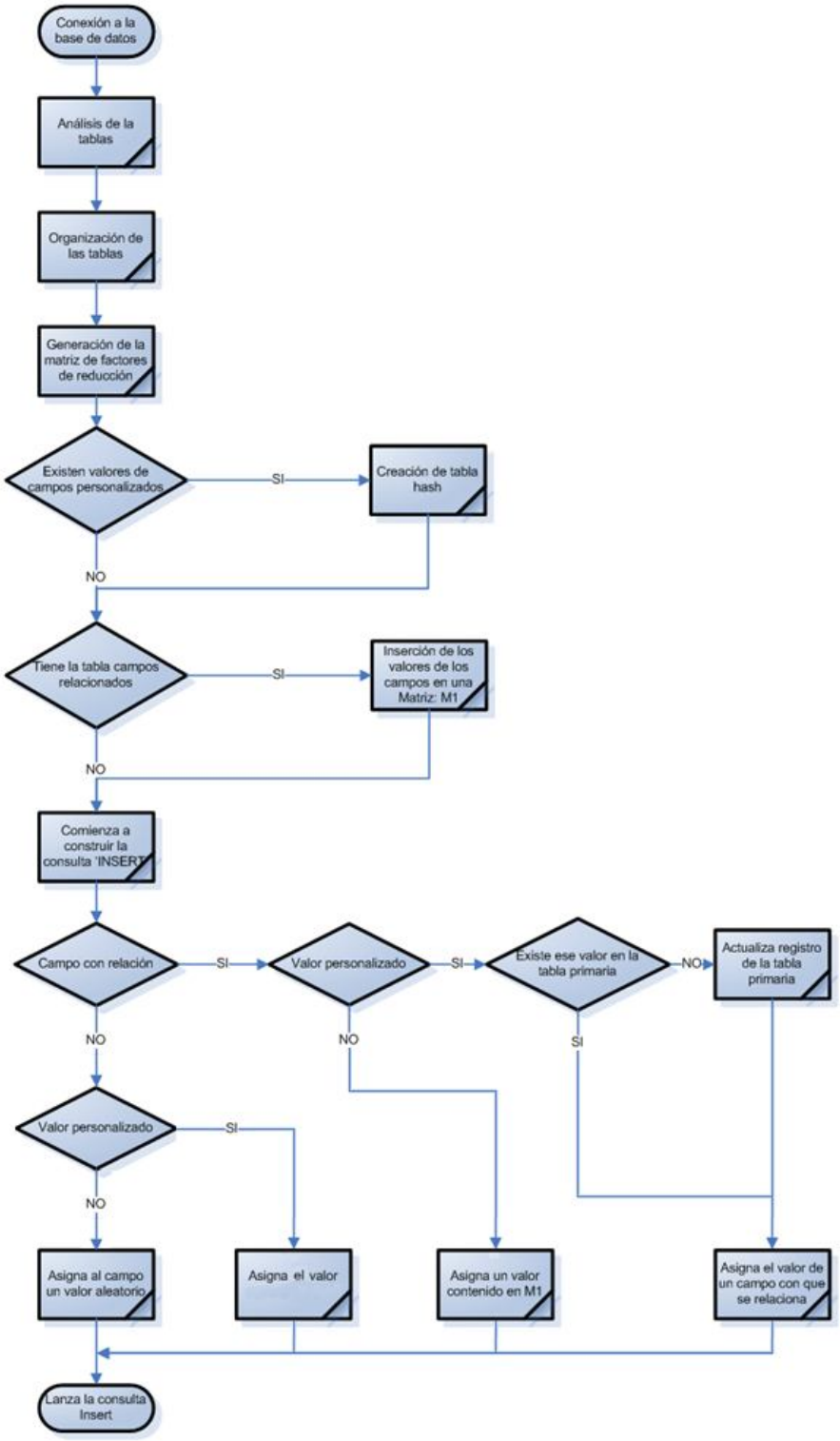


Figura 4.3.12: Diagrama de flujo de inserción de registros.

- Process: se trata de la clase encargada del proceso de medición de rendimiento. Dado que tiene que realizar numerosos análisis y cálculos se ha dividido algunas tareas en métodos.
    - actionPerformed: gestiona todos los eventos sobre los diferentes elementos contenidos en el panel.
    - calcularFactorBase: nos proporciona a partir del nombre de un SGBD un código interno.
    - calcularNumeroCampos: calcula el número de columnas con contenido de la fila de una matriz.
    - calcularNumeroTablas: calcula la posición del nombre de una tabla en una fila de una matriz.
    - formatoTiempo: genera un texto con un formato temporal adecuado a la aplicación.
    - restaurarIndices: restaura la clave primaria, relaciones o índices de una tabla, para ello recibe el nombre de la tabla y la información de las tablas de la base de datos, este método tiene su origen en la perdida de índices que se puede producir al copiar tablas. Devuelve un código que informa acerca del resultado de la operación.
    - run: se trata de un método para crear un hilo de ejecución con baja prioridad e irá insertando los registros en la base de datos y en caso de bloqueos o errores permitir al usuario pausar o detener el proceso de inserción.
    - tablaFactor: calcula un factor que nos permite rellenar tablas teniendo en cuenta sus relaciones y mantener una semejanza de tamaño entre diferentes patrones de mapeo y de este modo poder realizar pruebas equivalentes.
    - tablaFactorHerenciaClaseTabla: calcula el factor para tablas que se utilizan en un patrón de mapeo *herencia una clase una tabla*.
    - tablaFactorHerenciaRutaTabla: calcula el factor para tablas que se utilizan en un patrón de mapeo *herencia una ruta una tabla*.
    - tablasRelacionadas: creará una estructura que nos permitirá conocer las relaciones existentes entre una tabla con el resto de tablas de una base de datos. Recibirá como argumentos el nombre de la tabla en cuestión, la información del todas las tablas y un bool que indicará el sentido de recorrido de las relaciones.
    - Visualizar: gestiona la visualización y acción de todos los elementos del panel.

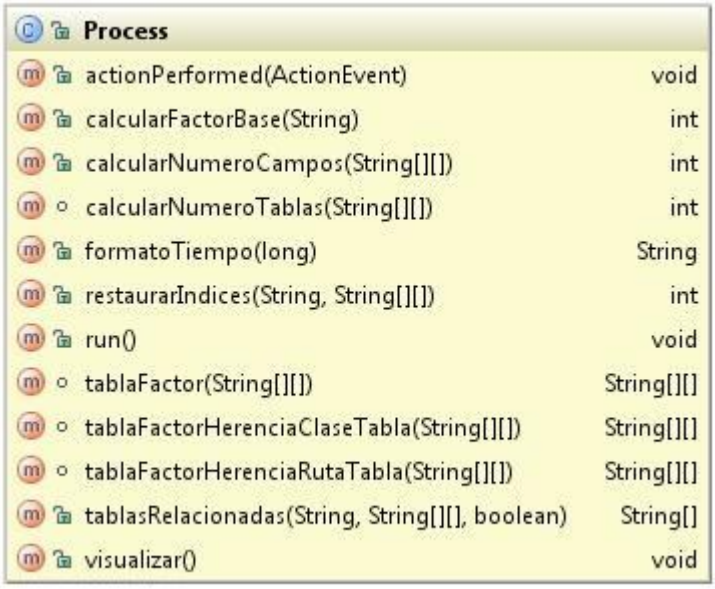


Figura 4.3.13: Clase Process.

El proceso de toma de medidas de rendimiento de una consulta puede parecer trivial, sin embargo se ha de tener en cuenta que para obtener un resultado que se ajuste a la realidad es necesario realizar la misma consulta un cierto número de veces y así poder valorar posibles variaciones debido a latencias de la red, procesos internos de un ordenador o servidor ajenos al proceso de medición ..., de modo que es importante que el estado de las tablas sea el mismo cada vez que se realiza la consulta, de modo que es necesario realizar una labor de restauración de las tablas al inicio de las consultas. En el siguiente diagrama de flujo podemos visualizar el diseño del proceso de medición.

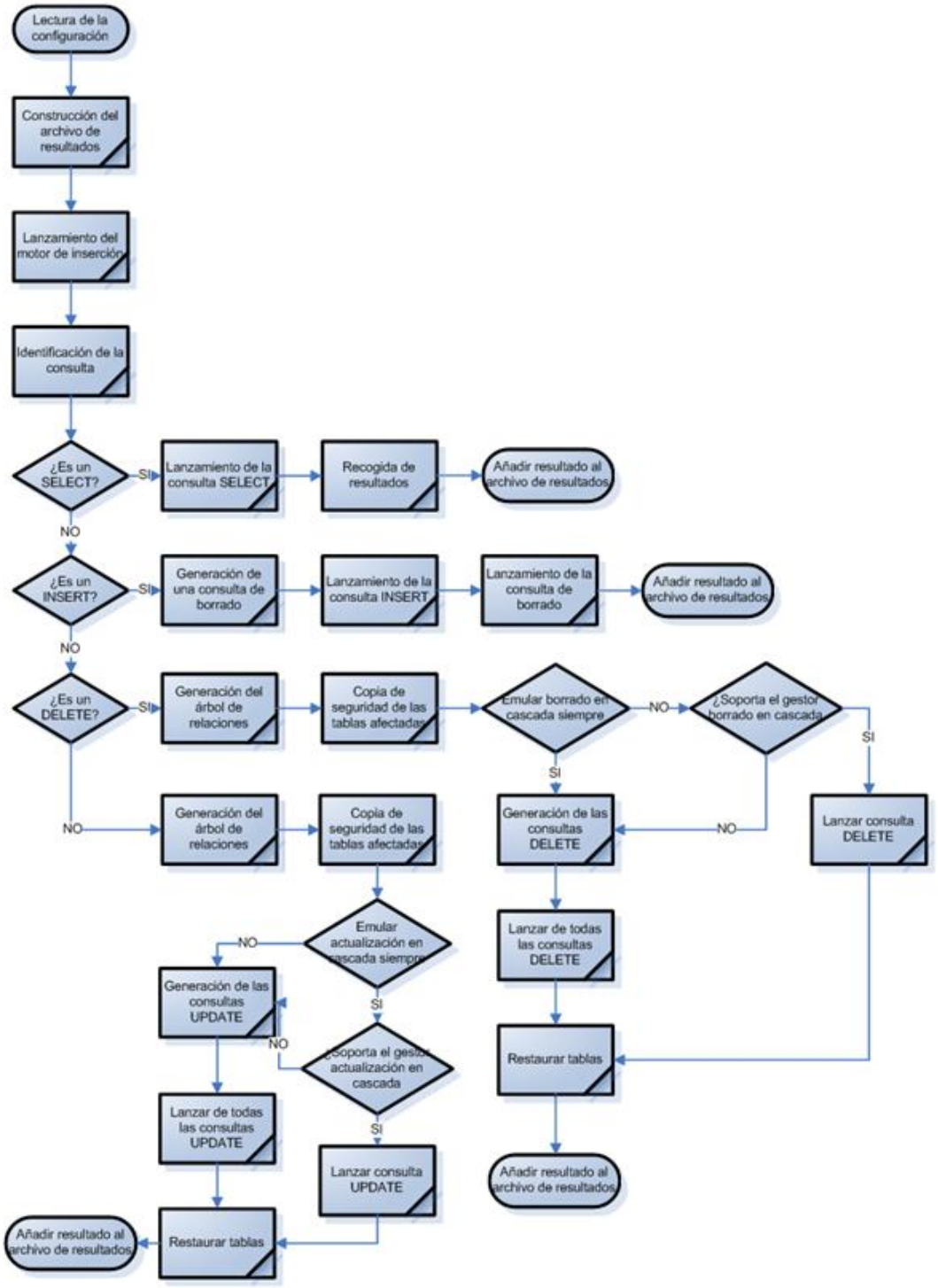


Figura 4.3.14: Diagrama de flujo de realización de mediciones.

Paquete Visualizer

Este paquete de clases es el encargado de realizar todas las tareas relativas a la visualización, análisis, comparación y almacenamiento de los resultados obtenidos en las pruebas de rendimiento. En la siguiente figura se puede observar las clases implicadas.

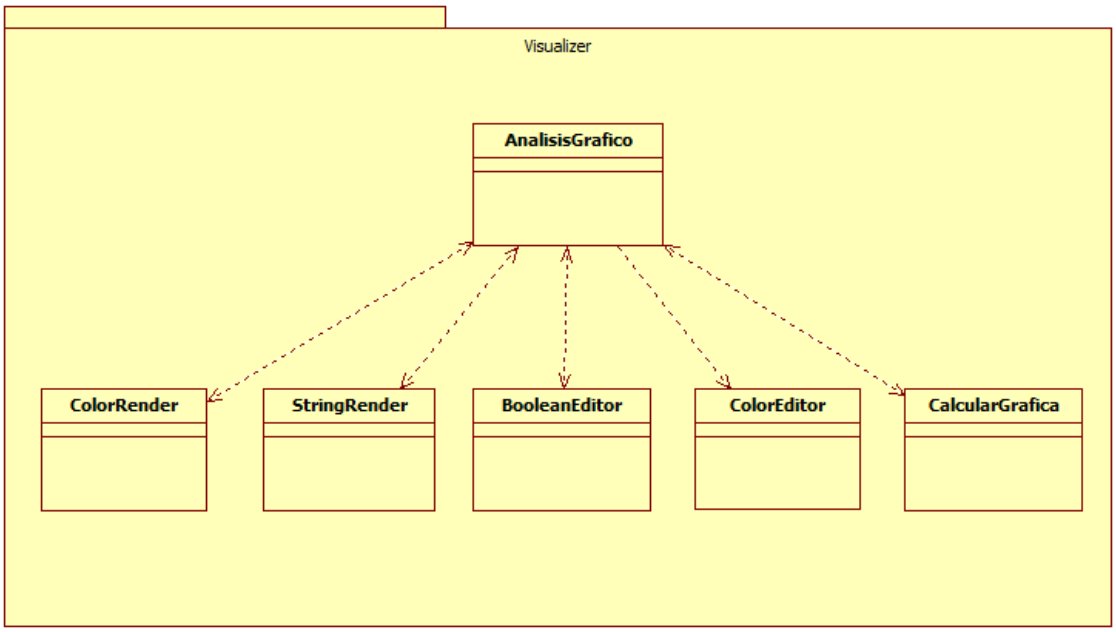


Figura 4.3.15: Paquete Visualizer.

- ColorRender: se encarga del renderizado de las celdas de color que indican los colores asignados al conjunto de resultados. Contiene únicamente un método.
    - getTableCellRendererComponent: recibe como argumentos los datos que ubican la celda y el color.

Figura 4.3.16: Clase ColorRender.

- StringRender: clase encargada del renderizado de textos emergentes en las tablas. Dado que durante la visualización de los resultados hay que mostrar al usuario información adicional y las consultas analizadas es necesario idear un sistema que permita ver esa información sin recargar las tablas, algunas consultas pueden estar compuestas por cientos de caracteres, de modo que se recurre al uso de texto emergente cuando el ratón se coloque encima de determinadas zonas de las tablas.
    - getTableCellRenderComponent: método encargado de mostrar el texto emergente, como argumentos se le proporciona toda la información necesaria que le indique sobre qué elemento está colocado el ratón.

Figura 4.3.17: Clase StringRender.

- BooleanEditor: clase que se encarga de dibujar y variar el valor de los checkbox en las tablas donde se visualizan los datos, de este modo se podrá seleccionar que datos se desean visualizar.
    - actionPerformed: gestiona todos los eventos sobre los checkbox.
    - getCellEditorValue: nos proporciona el estado de los checkbox.
    - getTableCellEditorComponent: dibuja el checkbox en la tabla, como argumentos recibe los datos que lo ubican en la tabla.

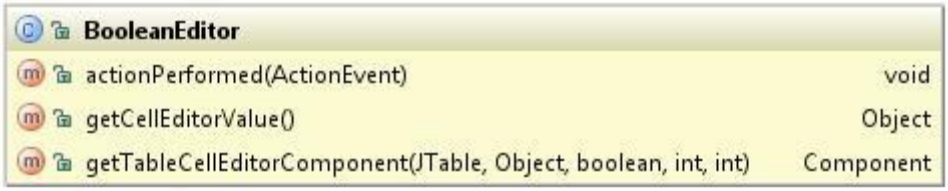


Figura 4.3.18: Clase BooleanEditor.

- ColorEditor: Clase que nos permite seleccionar el color de los diferentes conjuntos de datos. Su constructor crea un pop-up donde se nos presenta un selector de colores.
    - actionPerformed: modifica el color de las celdas en función del color seleccionado.

Figura 4.3.19: Clase ColorEditor.

- CalcularGrafica: Procesa todos los datos y genera una matriz con todos los resultados y para diferentes tipos de estadísticos.
    - Calcular: calcula el estadístico deseado que se proporciona como un argumento de entrada codificado.

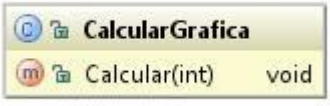


Figura 4.3.20: Clase CalcularGrafica.

- AnalisisGrafico: Se trata de la clase central para la visualización, análisis y almacenado de los resultados obtenidos en las pruebas.

- actionPerformed: gestiona todos los eventos sobre los diferentes elementos contenidos en el panel.
- decodificar: se encarga de decodificar los ficheros de texto y almacenarlos en memoria en un formato legible para el usuario.
- salvar: guarda los datos de las graficas. Nos muestra un pop-up donde seleccionar ubicación y nombre del fichero que se va a generar.
- visualizar: panel donde se ubican todos los elementos necesarios para la visualización, análisis y almacenamiento de los datos. Constará de una región donde se mostrará una gráfica o una tabla con los resultados de las pruebas, un selector de estadísticos, selector de escala, checkbox para el cálculo aproximado de valores entre muestras, una tabla que permitirá visualizar las consultas de creación de tablas y las consultas objeto de medición, checkbox de visualización y de colores y por ultimo un área de botones para la gestión de los ficheros, añadir, cargar y salvar.

Figura 4.3.21: Clase AnalisisGrafico.

5. Implementación

5.1 Introducción

Se trata de una herramienta que obtiene y analiza los tiempos de respuesta de diferentes sistemas gestores a determinadas consultas SQL. Esta herramienta está integrada en otra que permite visualizar gráficamente la estructura y las relaciones entre las tablas creadas por el usuario en un sistema gestor de bases de datos.

Las tablas sobre las que se van a realizar las consultas pueden ser definidas por el usuario o predefinidas de manera automática por la herramienta. Al igual que sucede con la definición de tablas, las consultas SQL también pueden ser definidas por el usuario o bien se pueden utilizar las consultas predefinidas en la herramienta. Los modos de realizar pruebas se dividen en:

- *Manual*: El usuario define las consultas de creación de tablas, consultas SQL sobre las que se realizarán las medidas, así como la configuración especial sobre registros personalizados y distribuciones concretas de registros en el mapeo de la herencia. Se trata de un modo ideal para la realización de diseños personalizados de bases de datos en función de las consultas SQL típicas que se van a utilizar.
- *Automático*: El usuario selecciona los mapeos y las consultas predefinidas por la herramienta. Este modo se utiliza para el estudio del rendimiento de los diferentes patrones de mapeos de objetos sobre bases de datos relacionales.

En ambos modos, la herramienta gestiona la inserción de registros en las tablas y la ejecución de las consultas en los sistemas gestores de bases de datos.

Tras la obtención de las medidas, los datos obtenidos pueden ser almacenados, exportados, combinados con resultados anteriores y visualizados en gráficas o mostrados en tablas.

5.2 Funcionamiento de la herramienta

La herramienta está integrada dentro de otra, ejecutándose al pulsar un botón y poniendo en uso un área de texto (ver figura 5.2.1). Pese a que la aplicación en la que ha sido integrada cuenta con un preprocesador que permite el análisis de la estructura de las tablas, se ha optado por programar un analizador nuevo totalmente diferente, debido a las limitaciones del anterior. En éste, las consultas de creación de las tablas han de ser escritas de una manera rígida que elimina la flexibilidad en la creación de tablas, es imposible el uso con sistemas gestores de bases de datos diferentes al motor JET, no analiza los índices de las tablas y presenta serios problemas para el uso intuitivo de la aplicación (debido a la necesidad de utilizar el sistema gestor de la base de datos para eliminar manualmente las tablas creadas tras cada análisis).

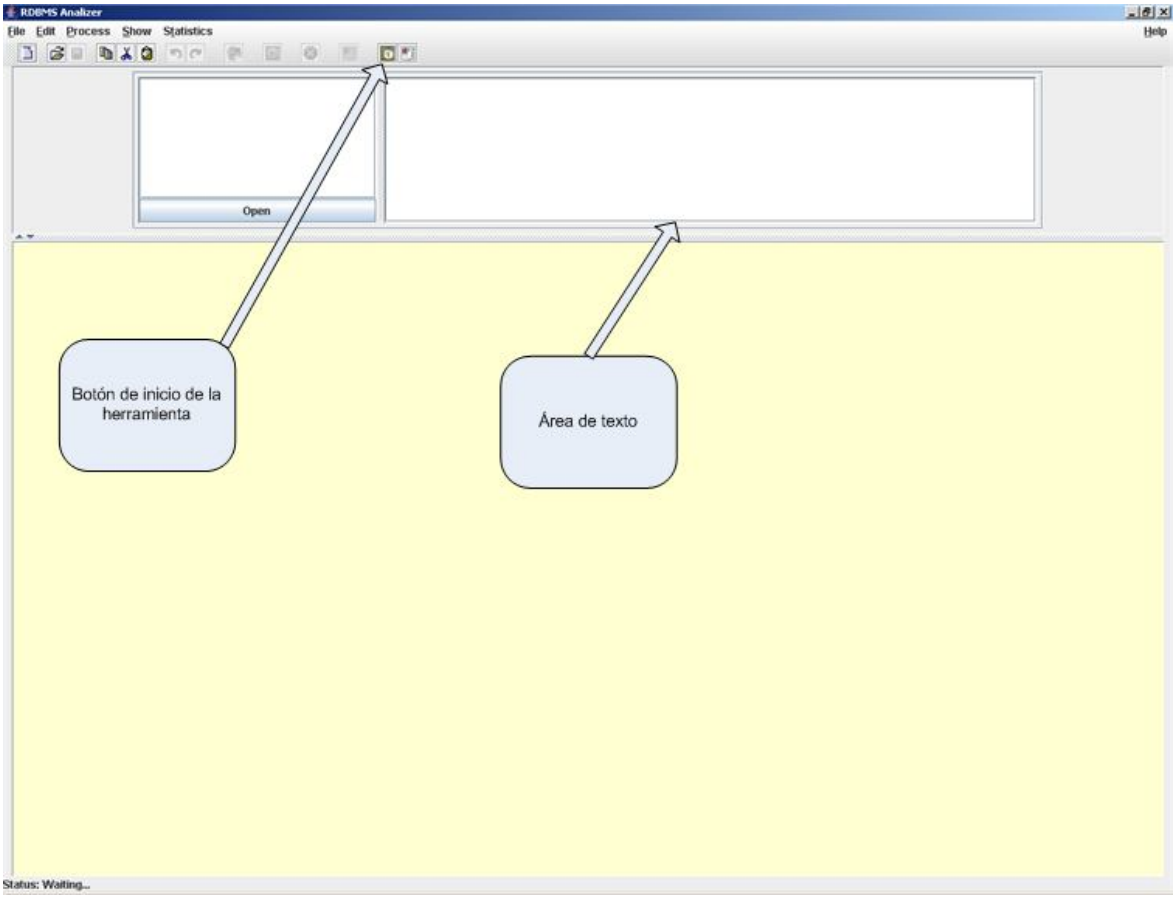


Figura 5.2.1: Ventana principal de la herramienta.

Las características básicas del funcionamiento de la herramienta se expresan en los siguientes apartados.

- Configuración y selección de mapeos.
- Generación de las bases de datos.
- Toma de medidas.
- Visualización de los resultados.

5.2.1 Configuración y selección de mapeos

El formulario principal de la aplicación (figura 5.2.2) cuenta con varias pestañas desde las que se pueden configurar los principales parámetros de la herramienta.

En la primera pestaña se encuentran los elementos que permiten seleccionar el tipo de medición a realizar (manual o automática), así como dos botones que permiten acceder a las ventanas de configuración de cada tipo de medición. En la figura 5.2.1.1 se puede observar dicha pestaña.

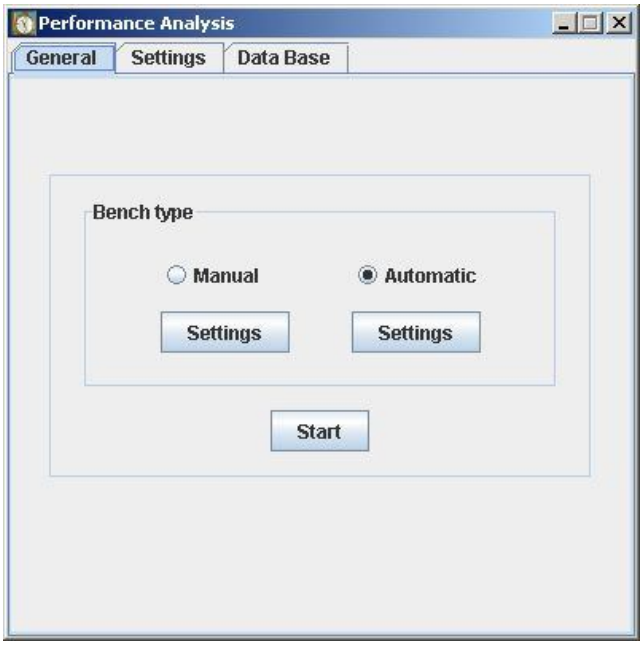


Figura 5.2.1.1: Ventana principal del módulo de medición.

- *Configuración de la medición manual*: Permite al usuario escribir las consultas que desea realizar sobre las tablas que previamente han sido definidas en el editor que se encuentra en la página principal de la herramienta, ver figura 5.2.1. Para ello dispone de un área de texto en la que se escriben tantas consultas como se desee. Antes de salir del formulario de configuración manual, se validan las consultas y en caso de que se incurra en algún tipo de error se informa al usuario del error y de la consulta que lo contiene. Para generar las consultas, el usuario puede necesitar información sobre las tablas y registros con que cuenta la base de datos. Para ello la herramienta muestra un análisis detallado de las tablas. Este análisis incluye: nombre de las tablas, nombre de los campos, tipo de dato, tamaño del tipo de dato, si es una clave principal, si se trata de un índice, caso de serlo el nombre del índice, si se trata de un índice único, si se trata de una clave extranjera y de tratarse de una clave extranjera muestra la tabla con que se relaciona, así como el campo por el que se relaciona. Dado que para determinadas consultas el usuario puede necesitar que ciertos registros contengan campos con un valor concreto, la herramienta añade una nueva columna en la tabla donde se expone el análisis de la base de datos para que el usuario pueda añadir el valor que desea que contengan ciertos registros. Si el usuario decide añadir registros con valor personalizado, también ha de especificar el número de registros con ese valor en concreto. Lo puede hacer seleccionando un porcentaje del total o un número absoluto de registros. La figura 5.2.1.2 muestra un ejemplo de la ventana de configuración manual. En ella se puede apreciar el análisis completo de las tablas sobre las que se van a realizar las medidas, el valor personalizado para un campo (El 25% de los registros de la tabla Clientes tendrán 'Pedro' en el campo Nombre) y las consultas SQL que el usuario ha decidido evaluar. Si el usuario quiere hacer uso de distribuciones especiales de los registros en las tablas, dispone de la posibilidad de seleccionar dos distribuciones especiales para la herencia: *una clase- una tabla* y *una ruta de herencia - una tabla* (en la sección del motor de inserción, se explica en detalle la función de estas opciones).

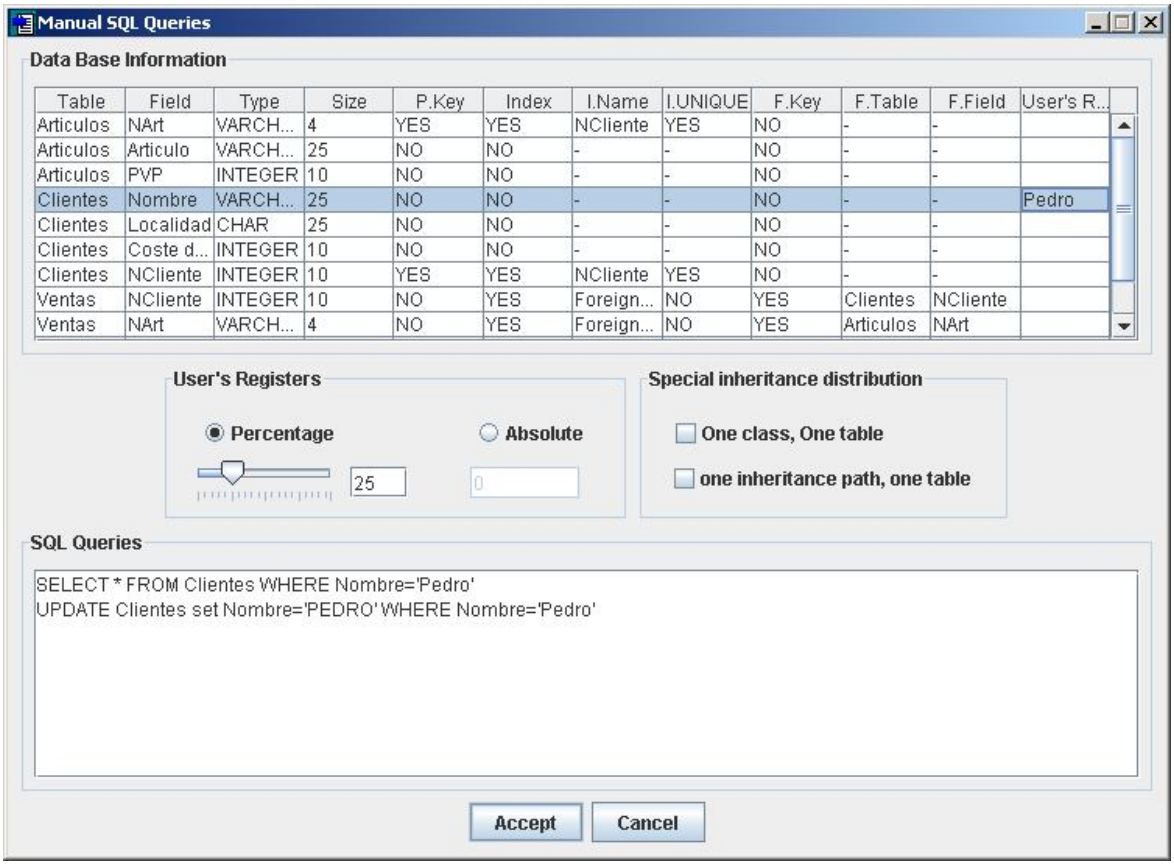


Figura 5.2.1.2: Panel de medición manual.

- *Configuración de la medición automática*: En esta configuración se realizan mediciones sobre patrones de mapeo y consultas predefinidas, tantas como se deseen. Para la selección de patrones el usuario cuenta con un panel (ver figura 5.2.1.3) donde puede visualizar gráficamente el mapeo que selecciona. La selección de las consultas predefinidas se realiza marcando las casillas de las consultas genéricas. Éstas se encuentran divididas en cuatro pestañas correspondientes a las sentencias SQL: *'select'*, *'insert'*, *'update'* y *'delete'*.

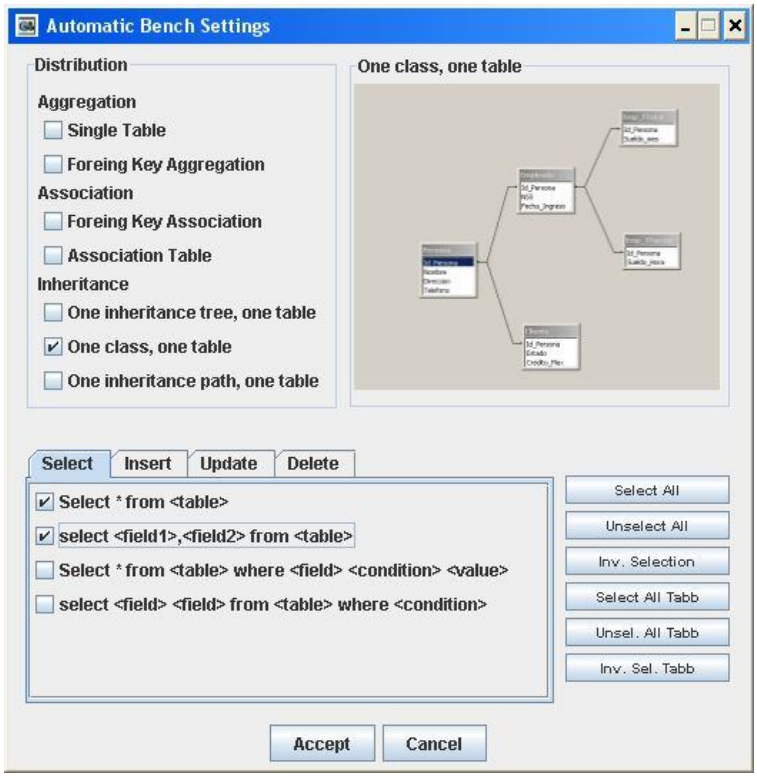


Figura 5.2.1.3: Panel de medición automática.

Pese a que se trata de una configuración automática sobre unos patrones de mapeo predefinidos y pese a que la herramienta cuenta con la posibilidad de realizar mediciones de consultas personalizadas (medición manual), los patrones de mapeo y las consultas predefinidas del modo automático pueden ser modificadas mediante la edición de una serie de archivos de texto.

- *Tablas*: Las consultas de creación de las tablas se encuentran almacenadas en los archivos de texto ubicados en la carpeta *tablas*. Estas se encuentran organizadas según se trate de agregación, asociación o herencia, y según el sistema gestor sobre el que van a trabajar. En la figura 5.2.1.4 se puede observar el contenido de uno de estos archivos de texto.

```
CREATE TABLE Director (id_Director INTEGER PRIMARY KEY, Apellidos VARCHAR(50),
Nombre VARCHAR(50), Empresa VARCHAR(50))
CREATE TABLE Empleado (id_Empleado VARCHAR(12) PRIMARY KEY,id_Director INTEGER,
Apellidos VARCHAR(50),Nombre VARCHAR(50), Fecha_Empleo DATE, FOREIGN KEY
(id_Director) REFERENCES Director (id_Director))
```

Figura 5.2.1.4: Contenido de fichero de creación de tablas.

- *Consultas:* Las consultas, objeto de medida, se encuentran almacenadas en los archivos de texto que se encuentran en la carpeta *Queries* y están organizadas en cuatro conjuntos: select, insert, update y delete. El formato de estos archivos ha de cumplir ciertas normas:

  - #: El carácter '#' permite añadir comentarios.
  - QXY: indica que se trata de una consulta.

- X: Tipo de mapeo.
  - 1: corresponde a un mapeo de agregación en tabla única.
  - 2: mapeo de agregación con clave extranjera.
  - 3: mapeo de asociación con clave extranjera.
  - 4: mapeo tabla de asociación.
  - 5: mapeo de árbol de herencia.
  - 6: mapeo de herencia una clase, una tabla.
  - 7: mapeo una ruta de herencia, una tabla.
- Y: Sistema gestor de bases de datos.
  - A: Access (Motor JET)
  - O: Oracle.
  - M: MySQL.
  - P: Postgres.

o Si se edita sin comentar (‘#’) y sin el modificador Q el programa escribirá la sentencia en las pestañas de consultas de la configuración automática. De este modo se pueden agrupar ciertas consultas, como un tipo genérico de consultas.

En la figura 5.2.1.5 se puede observar un fragmento de un archivo de configuración de consultas predefinidas.

```
#
#     DELETE
#
#
#In this file you can add select queries
#
#if you want write a comment you can use the character '#'
#You have to write first the sencentece that user can read in autor
#with the prefix Q<mapped>
#     example:
#           select * from <table> where <condiction>
#           Q1 select * from clients where name='John'
#           Q2 select * from vendors where country='Spain'
#It is very important that SQL Query is correct
#
# Q1->Aggregatin Single Table
# Q2->Aggregation Foreign Key Agregation
# Q3->Association Foreign Key Association
# Q4->Association Association Table
# Q5->Inheritance, one inheritance tree, one table
# Q6->Inheritance, one class, one table
# Q7->Inheritance, one inheritance path, one table
#
#
#
#  Data Base
#  A->MS ACCESS
#  O->Oracle
#  M->MySQL
#  P->Postgress
#
#
#

Delete * from <table>

Q1A DELETE FROM Clientes
Q1O DELETE FROM Clientes
Q1M DELETE FROM Clientes
Q1P DELETE FROM Clientes


#Q2 DELETE FROM Cliente
Q2A DELETE FROM Tipo_Direccion
Q2O DELETE FROM Tipo_Direccion
Q2M DELETE FROM Tipo_Direccion
Q2P DELETE FROM Tipo_Direccion
```

Figura 5.2.1.5: Contenido de fichero de consultas objeto de medición.

- *Especiales:* En este archivo se encuentra la configuración de ciertos parámetros especiales. Se encuentra en la carpeta *Queries* y su nombre es *Especiales.txt*. Al igual que los archivos de las consultas, cuenta con un formato concreto:

    - #: Permite añadir comentarios.
    - QX: especifica el tipo de mapeo, los valores de la 'X' son los mismos que para el caso de los archivos de configuración de consultas predefinidas.
    - *Datos especiales*: tras el modificador 'QX' se han de escribir los parámetros especiales, con el siguiente formato N$%&V$%&U$%&M, donde:
        - $%&: separador de variables.
        - N: número del campo que contendrá un valor personalizado.
        - V: valor del campo personalizado.
        - U: porcentaje o valor absoluto de registros que contendrán el valor personalizado.
            - *Percentage of user's registers is X%:* donde 'X' corresponde al porcentaje de registros que contendrán el valor personalizado.
            - *Number of user's registers is Y*: donde 'Y' indica el número absoluto de registros que contendrán el valor personalizado.
        - M: tipo de mapeo.

En la figura 5.2.1.6 se puede observar un ejemplo de un archivo especial.

```
# Special registers
#
# Formar
#   Q<map type> <number of field>$%&<value>$%&<special string, number of special registers>$%&<special information>
#
# Q1->Aggregatin Single Table
# Q2->Aggregation Foreign Key Agregation
# Q3->Association Foreign Key Association
# Q4->Association Association Table
# Q5->Inheritance, one inheritance tree, one table
# Q6->Inheritance, one class, one table
# Q7->Inheritance, one inheritance path, one table
#


Q1 4$%&Mayor$%&Percentage of user's registers is 30%$%&Mapped Aggregation : Single Table
Q2 1$%&Mayor$%&Percentage of user's registers is 30%$%&Mapped Aggregation : Foreign Key
Q3 3$%&Telefonica$%&Percentage of user's registers is 30%$%&Mapped Association : Foreign Key
Q4 3$%&Telefonica$%&Percentage of user's registers is 30%$%&Mapped Association : Table Association
Q5 9$%&1234567891$%&Percentage of user's registers is 12%$%&Mapped Inheritance : one inheritance tree, one table
Q6 13$%&1234567891$%&Percentage of user's registers is 60%$%&Mapped Inheritance : one class, one table
Q7 19$%&1234567891$%&Percentage of user's registers is 60%$%&Mapped Inheritance : one inheritance path, one table
```

Figura 5.2.1.6: Contenido del fichero de datos especiales.

En la segunda pestaña de la ventana principal (ver figura 5.1.9) se encuentran los elementos que permiten configurar los parámetros comunes a ambos tipos de medición. Los parámetros a disposición del usuario son:

- *Número de registros por tabla*: Se seleccionan marcando las casillas que correspondan con el número de registros por tabla que se desean generar. Pueden ser marcados varios a la vez.
- *Mostrar los resultados de las consultas*: Si el usuario lo desea, puede marcar esa casilla de modo que una vez se haya realizado la medición de

rendimiento sobre la base de datos, pueda disponer de los resultados obtenidos por las consultas realizadas. Ver figura 5.2.1.7.

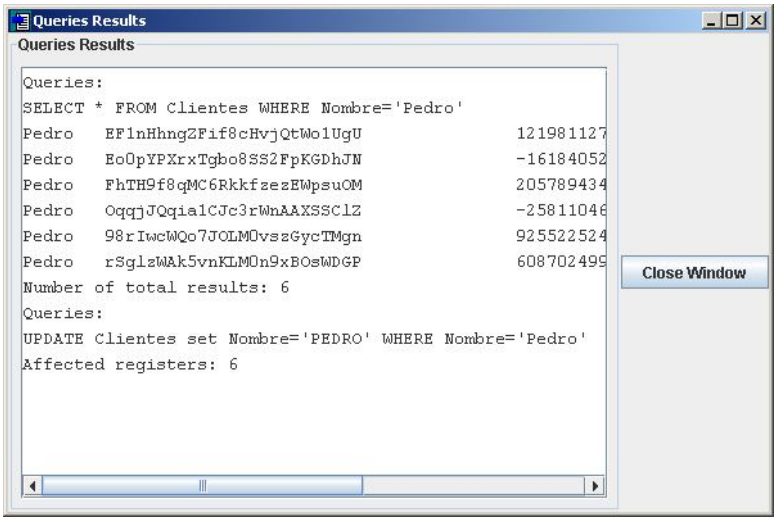


Figura 5.2.1.7: Panel de resultados de consultas.

- *Número de repeticiones*: Dado que se van a tomar mediciones de tiempo sobre consultas a bases de datos, es importante realizar varias medidas para de este modo poder realizar un trabajo estadístico sobre los resultados. El usuario puede variar el número de medidas que se van a realizar sobre la misma consulta. Como es de esperar, se obtienen resultados más significativos cuanto mayor número de tomas se realicen.

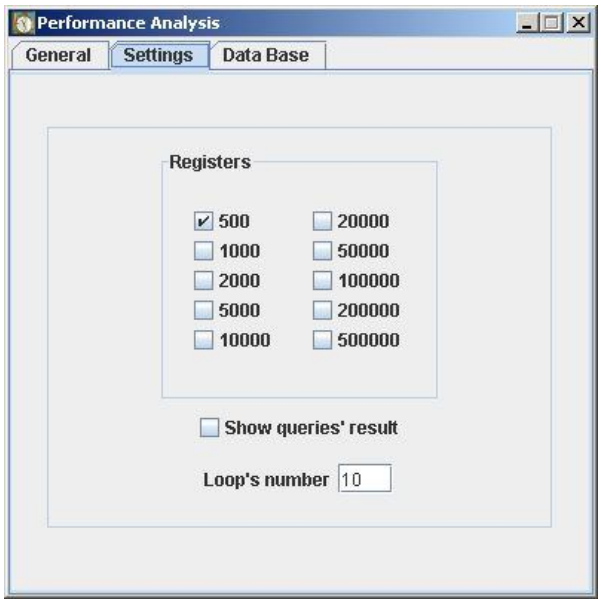


Figura 5.2.1.8: Panel de configuraciones básicas.

En la tercera pestaña (ver figura 5.2.1.9), se encuentran los elementos de configuración de los *drivers* de las bases de datos, así como la configuración del comportamiento del motor que gestiona las consultas a realizar.

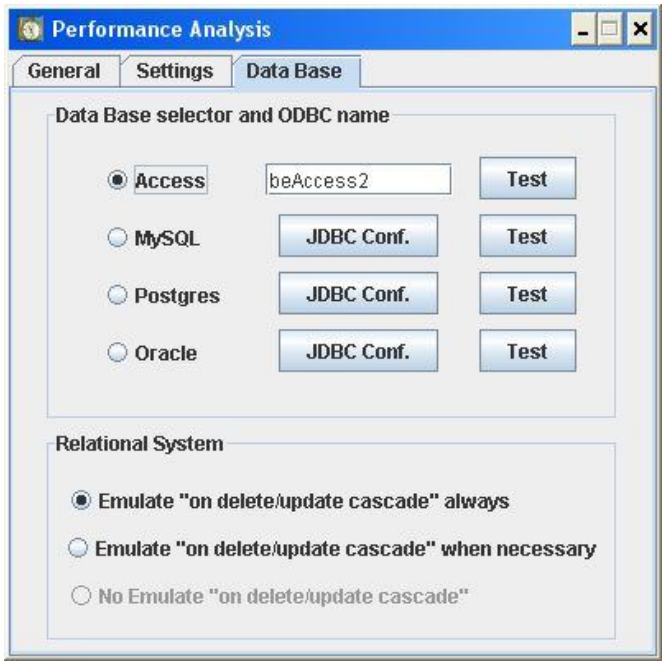


Figura 5.2.1.9: Panel de configuración de conectores.

- *Configuración de drivers*: Como se puede observar en la figura 5.2.1.9, el *driver* de *Access* se configura de una manera totalmente diferente al resto. Esto es debido a la no disponibilidad de *drivers* JDBC [14] para el motor JET de Microsoft [11], disponiendo tan sólo del *driver* ODBC [15]. Este *driver* se ha de configurar desde el panel de control del sistema operativo y desde la herramienta se modifica el nombre del *driver* para acceder a una base de datos basada en el motor JET. El resto de bases de datos disponen de *driver* JDBC, por ello desde la propia herramienta se pueden configurar todos los parámetros principales de éstos, como son el *host*, el puerto, el nombre de la base de datos, así como el usuario y la contraseña. Como ejemplo, en la figura 5.2.1.10 se puede observar la ventana de configuración del *driver* JDBC del gestor de bases de datos Oracle.

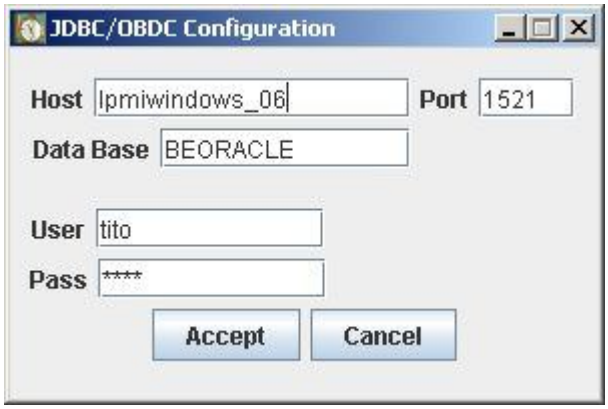


Figura 5.2.1.10: Ventana de configuración JDBC.

- *Comportamiento del motor de consultas*: El usuario tiene la posibilidad de seleccionar el tipo de actualización y borrado que desea que se realice cuando existen relaciones en las tablas. *Borrado y actualización en cascada* gestionado por la base de datos o *borrado y actualización en cascada emulado por el motor* que gestiona las consultas de la base de datos. En este caso, se analizan los caminos de las relaciones de las tablas implicadas en la actualización o borrado en cascada y el propio motor genera las consultas necesarias para que la actualización y el borrado sea en cascada y evitar los posibles errores que se pueden producir al violar la integridad referencial de la base de datos. También existe una tercera posibilidad que es no emular el borrado en cascada (esta posibilidad ha sido desactivada debido a los posibles problemas que puede provocar el violar la integridad referencial de las base de datos).

5.2.2 Generación de las tablas

El proceso global de la generación de las tablas en la base de datos sobre la que se van a realizar las mediciones cuenta con los siguientes puntos básicos:

- *Creación de las tablas.*
  - Mediante las instrucciones de creación de tablas, ya sean predefinidas o introducidas por el usuario, el programa crea las tablas en la base de datos.

- *Análisis de la base de datos.*
  - El programa analiza la estructura de la base de datos: nombre de las tablas, nombre de los campos, tipo de dato, tamaño del tipo de dato, si es una clave principal, si se trata de un índice (caso de serlo el nombre del índice, si se trata de un índice único) y si se trata de una clave extranjera (si es clave extranjera informa de la tabla a la que pertenece).

- *Generación de registros.*
  - Con la información recogida se van generando registros, teniendo en cuenta que los tipos y tamaños de los campos coincidan, no repitiendo claves principales y respetando la integridad referencial. Los valores que se asignan a los campos son aleatorios salvo cuando se trabaja sobre un campo con valor predefinido por el usuario.

- *Inserción de registros.*
  - Se genera la consulta de inserción y se van introduciendo los registros en las tablas hasta alcanzar el número de registros que determina el usuario.

5.2.3 Toma de medidas

Una vez se han generado las tablas de la base de datos con un número determinado de registros, se realizan sobre ella las consultas generadas por el usuario, almacenando en memoria el tiempo que tarda en ejecutarse cada una de ellas. Finalizada cada consulta, los

valores de los registros son restaurados a su valor inicial para evitar que se produzcan resultados no deseados. Las figuras 5.2.3.1 y 5.2.3.2 muestran la interfaz de usuario de la generación de las tablas, así como los resultados parciales de las medidas.

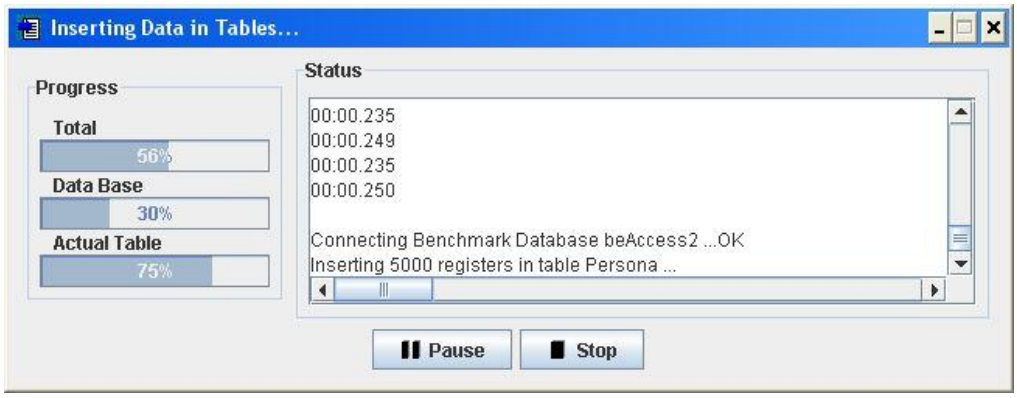


Figura 5.2.3.1: Ventana de proceso de inserción de registros en acción.

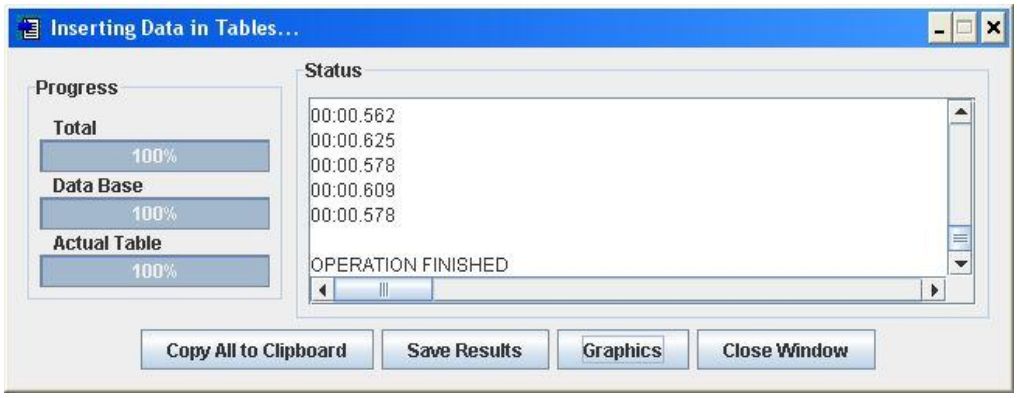


Figura 5.2.3.2: Ventana de proceso de inserción de registros finalizado.

5.2.4 Visualización de medidas

Una vez finalizado el proceso de toma de medidas, la herramienta permite al usuario visualizar los resultados obtenidos, guardar los resultados en fichero y cargar o añadir resultados anteriores. Los datos pueden ser presentados en una gráfica, o sí se desea en una tabla, visualizándose sólo los que desea el usuario. Como medidas estadísticas, la herramienta presenta: la *media*, la *media aritmética*, *la mediana* y la *desviación estándar*. Las figuras 5.2.4.1, 5.2.4.2 y 5.2.4.3 muestran la interfaz de usuario de visualización de las mediciones. La figura 5.2.4.1 muestra la gráfica (la mediana) de los resultados obtenidos al realizar diversas consultas sobre el sistema gestor MS Access. Como se puede observar el eje que hace referencia al número de registros se encuentra en escala logarítmica y no se dibujan todas las consultas sino sólo las deseadas por el usuario.

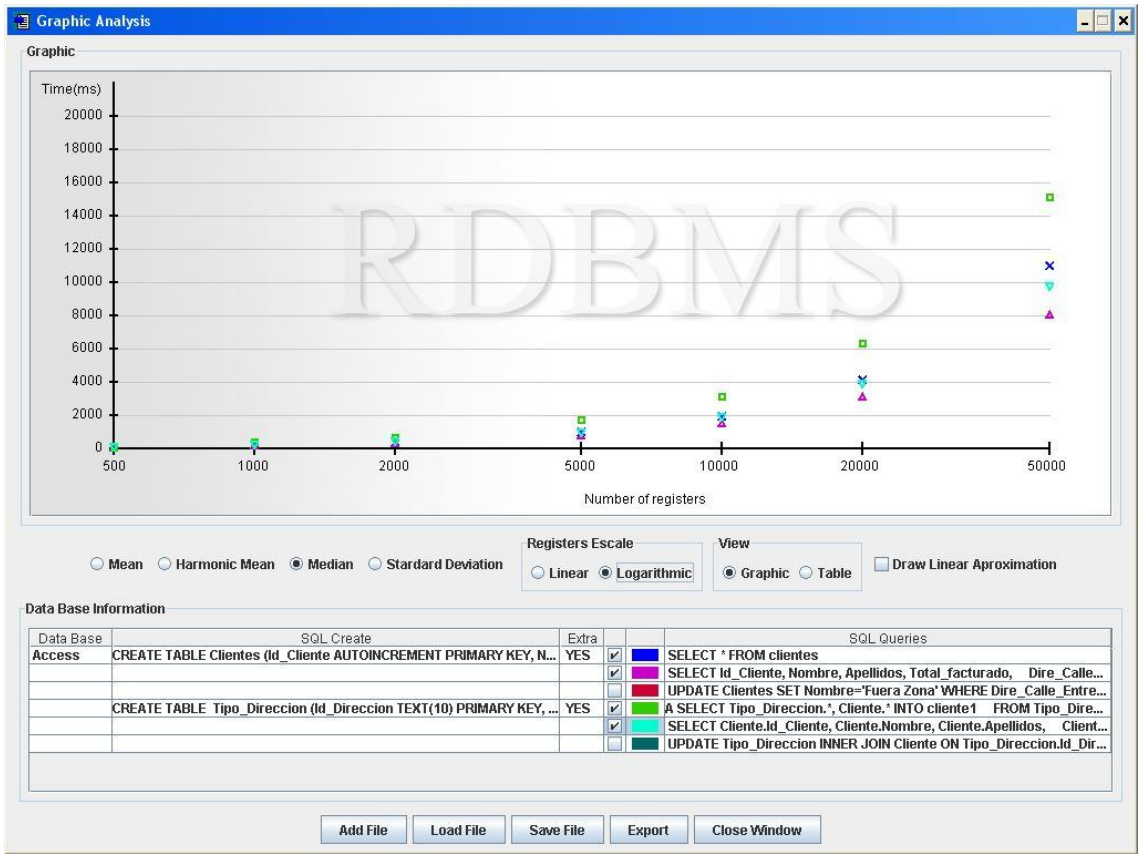


Figura 5.2.4.1: Ventana de visualización de resultados.

En la figura 5.2.4.2 se muestran las mismas medidas que las vistas en la figura 5.2.4.1, pero se aprecia el resultado de dibujar la aproximación lineal de las medidas para los intervalos entre el número de registros. También se aprecia la capacidad del programa de mostrar textos emergentes que aportan información sobre el gestor de base de datos, driver, fecha…etc. así como el texto completo de las consultas SQL. También es posible cambiar el color de los puntos y las líneas de aproximación de la gráfica correspondiente a cada sentencia SQL que se está analizando (ver figura 5.2.4.3) personalizando de esta manera el interfaz gráfico.

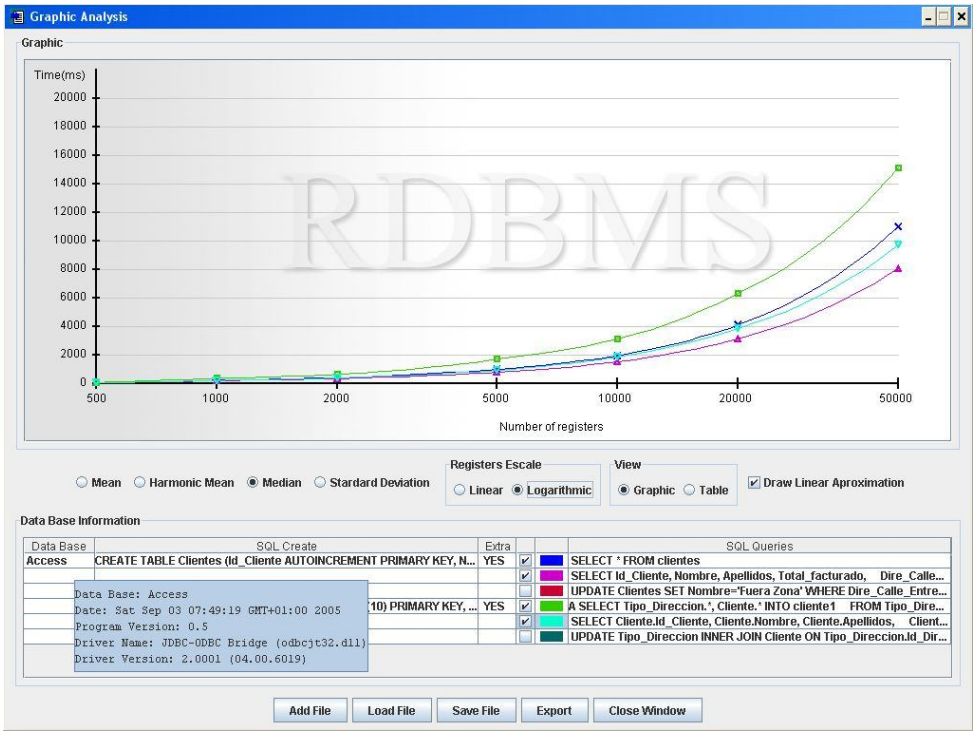


Figura 5.2.4.2: Ventana de visualización de resultados con aproximación.

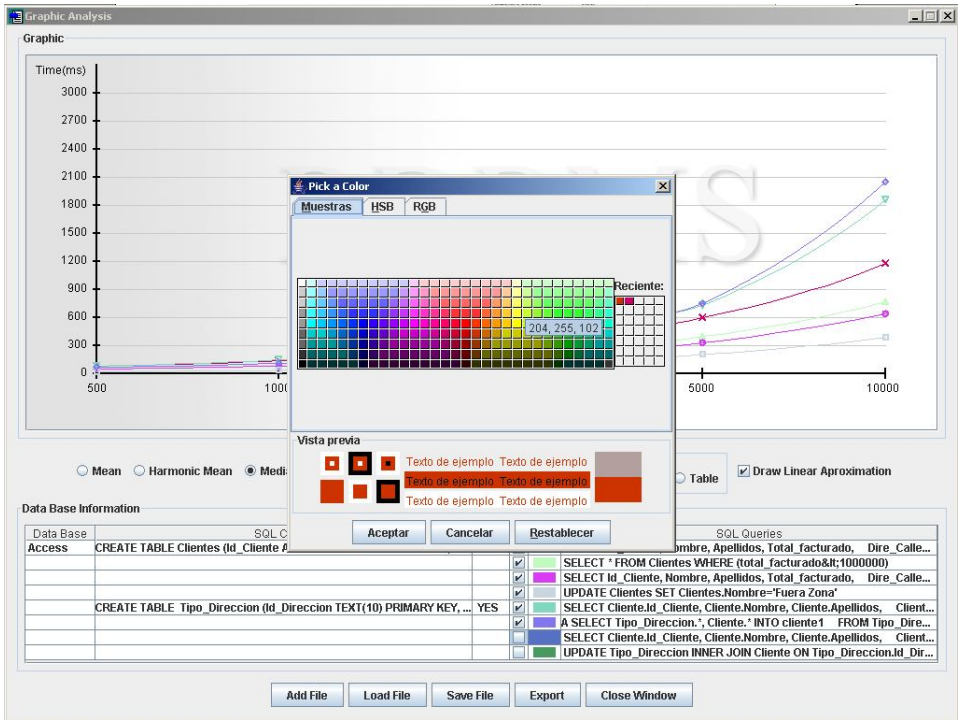


Figura 5.2.4.3: Visualización con colores.

Finalmente, la figura 5.2.4.4 muestra la media de los resultados obtenidos en forma de tabla.

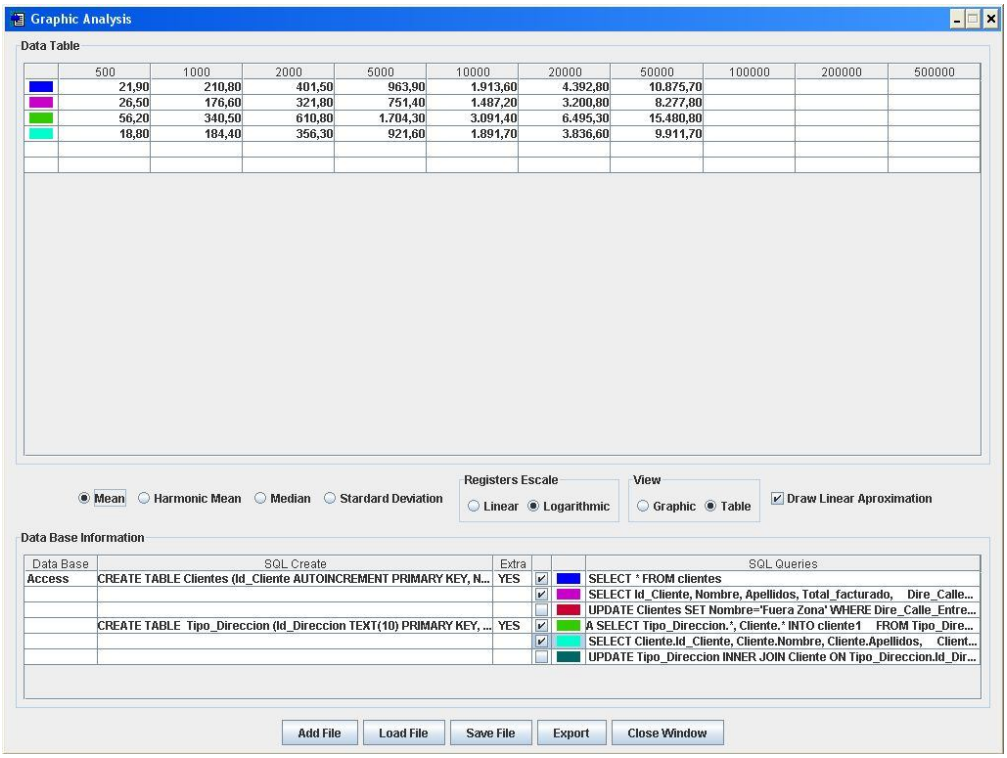


Figura 5.2.4.4: Visualización de resultados en tabla.

5.2.5 Descripción del motor de inserción de registros

Su principal función es la de insertar un número concreto de registros en las tablas de una base de datos realizando las siguientes tareas:

- *Analizar todas las tablas contenidas en la base de datos.*
  - Nombre de las tablas.
  - Nombre, tipo y tamaño de los campos.
  - Clave Primaria.
  - Índices.
  - Clave extranjera (tabla y campo primario).

- *Ordenar las tablas* de modo que sean las tablas primarias las que se sitúen en primera posición para la inserción de registros y así evitar errores debidos a la integridad referencial.

La figura 5.2.5.1 muestra el resultado obtenido por la aplicación tras analizar y ordenar las tablas.

Data Base Information

Table	Field	Type	Size	P.Key	Index	I.Name	I.UNIQUE	F.Key	F.Table	F.Field	User's R...
Persona	Id_Pers...	COUNT...	10	YES	YES	Index_5...	YES	NO	-	-	
Persona	Nombre	VARCH...	50	NO	NO	-	-	NO	-	-	
Persona	Direccion	VARCH...	50	NO	NO	-	-	NO	-	-	
Persona	Telefono	VARCH...	15	NO	NO	-	-	NO	-	-	
Cliente	Id_Pers...	COUNT...	10	YES	YES	Rel_78...	NO	YES	Persona	Id_Pers...	
Cliente	Estado	BIT	1	NO	NO	-	-	NO	-	-	
Cliente	Credito...	INTEGER	10	NO	NO	-	-	NO	-	-	
Emplea...	Id_Pers...	COUNT...	10	YES	YES	Rel_B2...	NO	YES	Persona	Id_Pers...	
Emplea...	NSS	VARCH...	30	NO	NO	-	-	NO	-	-	

Figura 5.2.5.1: Panel de información sobre tablas.

- *Generar una matriz* (*matriz de factores de reducción*) donde se asigna a cada tabla un número que informa sobre el número de registros que contendrá la tabla (*factor de reducción*) en función del número de registros que ha seleccionado el usuario. Esto permite controlar la distribución entre las tablas de los registros que se van a insertar y con ello poder obtener entornos de medida que permitan la comparación directa de los resultados obtenidos en las mediciones. El número de registros que van a ser insertados en la tabla es igual a la inversa del factor de reducción multiplicada por el número de registros seleccionado por el usuario.
  Esta matriz de factores de reducción adquiere especial relevancia en al caso de los patrones de mapeo para la herencia. Para describir de modo ilustrativo la generación de esta matriz se va a emplear un ejemplo de mapeo de herencia. El mapeo de herencia sobre el que se va a trabajar es el ilustrado en la figura 5.2.5.2

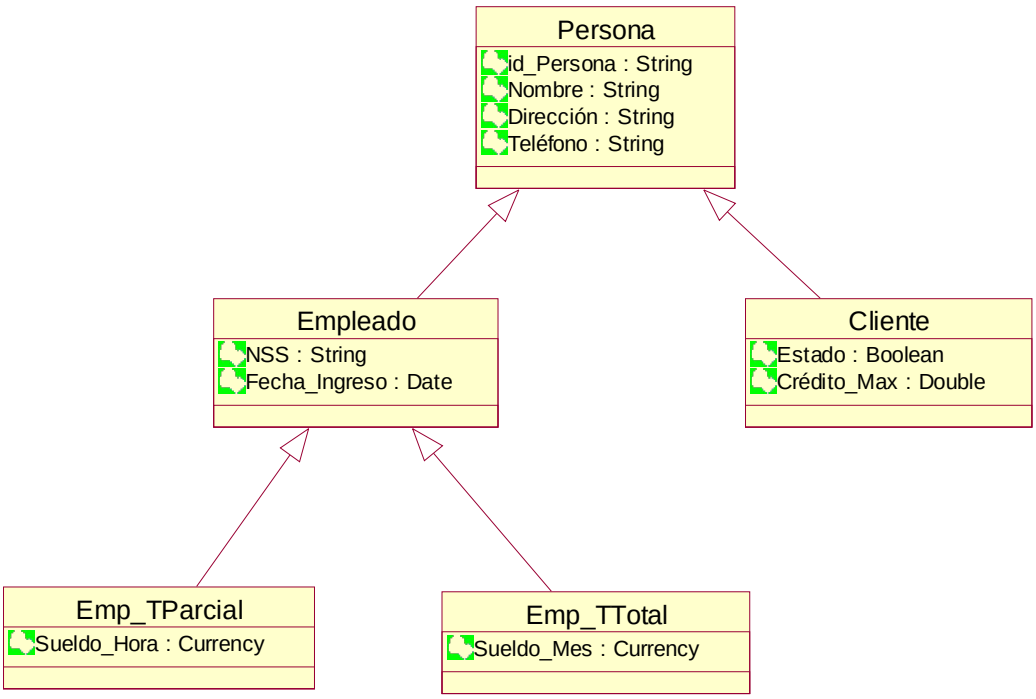


Figura 5.2.5.2: Ejemplo de mapeo de herencia.

Para mapear la herencia de la figura 5.2.5.2 empleando el patrón de mapeo un *árbol de herencia - una tabla* se ha de crear una tabla como la que aparece en la figura 5.2.5.3.

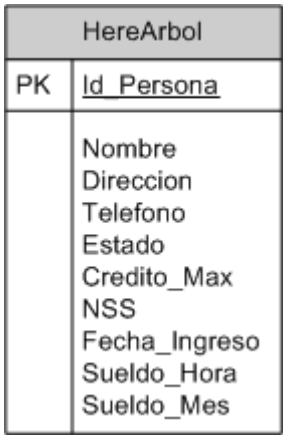


Figura 5.2.5.3: Un árbol de herencia una tabla.

Esta tabla contiene mapeados tantos objetos persistentes como registros se han insertado en ella, de modo que si se insertan 5000 registros significa que las medidas de tiempo sobre esta tabla se corresponden con las medidas realizadas sobre 5000 objetos. Por ello, para mantener la coherencia de las medidas tomadas sobre diferentes patrones de herencia, es necesario que se realicen sobre el mismo número de objetos y con la misma distribución.

Para mapear la herencia descrita en la figura 5.2.5.2 utilizando el patrón de mapeo *una clase - una tabla*, se han de generar las tablas descritas en la figura 5.2.5.4

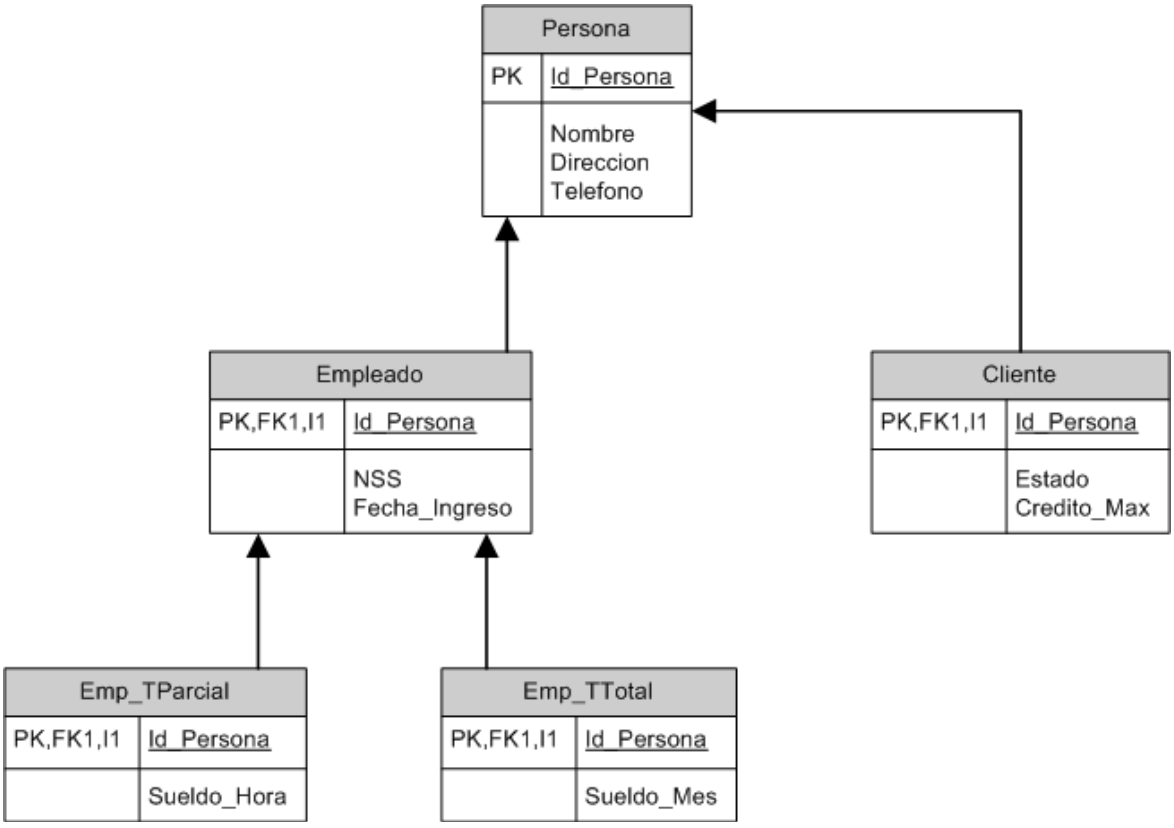


Figura 5.2.5.4: Una clase, una tabla.

Si se insertan en cada una de las tablas 5000 registros (hay que tener en cuenta que se ha de respetar la integridad referencial) significa que se está trabajando sobre un total de 15000 objetos, 5000 objetos corresponderían a la clase Cliente, 5000 a la clase Emp_TParcial, 5000 a la clase Emp_TTotal y las clases Persona y Empleado no contendrían ningún objeto. De modo que sí se desea poder comparar este patrón de mapeo con el patrón de mapeo *un árbol de herencia - una tabla* en igualdad de condiciones, se ha de realizar una distribución concreta de los registros.

Se ha optado por realizar una distribución uniforme que consiga almacenar en cada tabla el mismo número de objetos, de modo que si se va a trabajar sobre 5000 objetos estos se van a distribuir del siguiente modo: 1000 objetos pertenecerán a la clase Persona, 1000 a la clase Cliente, 1000 a la clase Empleado, 1000 a la clase Emp_TParcial y 1000 a la clase Emp_TTotal. Para obtener esta distribución de objetos se han de insertar 1000 registros en la tabla *Emp_TParcial*, 1000 en la tabla *Emp_TTotal*, 3000 registros en la tabla *Empleado* (1000 de estos registros corresponden a los 1000 objetos de la clase Emp_TTotal, 1000 a la clase Emp_TParcial y 1000 a la clase Empleado). En la tabla *Cliente* se insertarán 1000 registros obteniendo así 1000 objetos en la clase Cliente. Finalmente se insertarán 5000 registros en la tabla *Persona* (1000 de esos registros corresponden a los 1000 objetos de la clase Emp_TParcial, 1000 para la clase Emp_TTotal, 1000 para la Clase Empleado, 1000 para la clase Cliente y finalmente 1000 registros para los 1000 objetos de la clase Persona). Para obtener una distribución de los objetos comparable a la obtenida mediante el mapeo de *árbol de herencia* se ha de insertar 5000 registros en la tabla *Persona*, 1000 en la tabla *Clientes*, 3000 en la tabla *Empleado*, 1000 en la tabla *Emp_TParcial* y 1000 en la tabla *Emp_TTotal*. Para la obtención de esta distribución de registros, la herramienta ha de contar con la información concerniente al

tipo de mapeo y así generar la matriz de factores conveniente, en el caso del patrón de mapeo para herencia *una clase - una tabla*, los factores de reducción se generan combinando el número de descendientes que tiene cada clase y el número de objetos que desea insertar el usuario. La fórmula resultante es la siguiente:

$$RIT = \frac{1}{DT+1} \cdot OD$$

RIT: Registros a insertar en la tabla.
DT: Número de descendientes.
OD: Objetos deseados por el usuario.

Para poder comparar los patrones anteriores con el patrón de herencia *una ruta - una tabla* también se ha de hacer uso de una matriz de factores propia de este tipo de mapeo. Si deseamos mapear las clases de la figura 5.2.5.2, se han de generar las tablas indicadas en la figura 5.2.5.5.

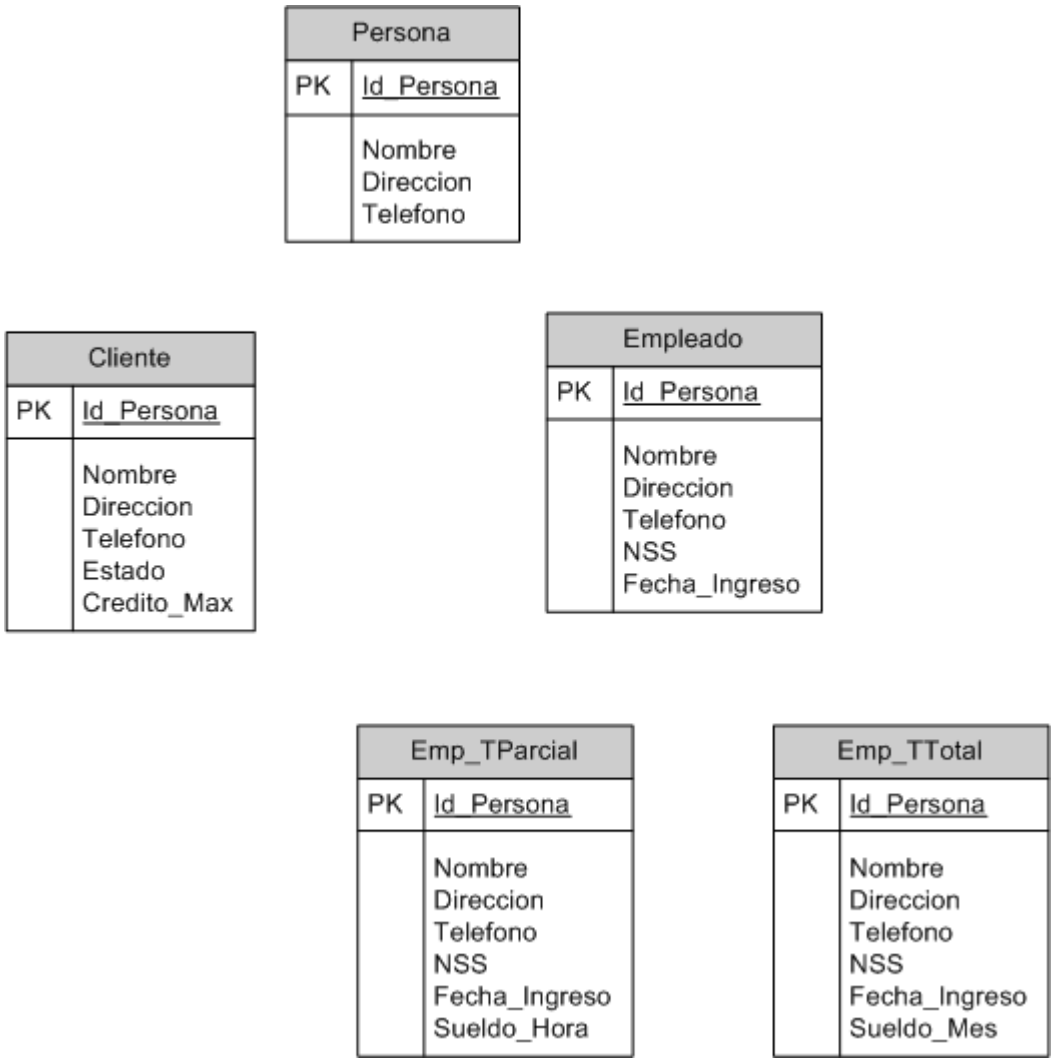


Figura 5.2.5.5: Una ruta - una tabla.

Si se insertaran 5000 registros por tabla, se obtendrían 25000 objetos mapeados, con lo cual no se estaría realizando una comparativa en igualdad de condiciones. Dado que para los casos anteriores se optó por una distribución uniforme de los objetos (1000 objetos por clase) en este caso también se han de realizar las medidas sobre 1000 objetos de cada clase. Para ello, se han de insertar 1000 registros en cada tabla y así obtener un total de 5000 objetos. Al igual que para el patrón de mapeo anterior, se ha de generar una matriz de factores específica para este patrón. Los factores se calculan según la siguiente fórmula:

$$RIT = \frac{OD}{NT}$$

RIT: Registros a insertar en la tabla.
OD: Objetos deseados por el usuario.
NT: Número de tablas.

Si el usuario desea asignar un valor concreto a uno de los campos de un número determinado de registros (porcentaje del total o cifra absoluta) se procede a la generación aleatoria de una tabla *hash* (no contiene valores repetidos) de números. Esta tabla tiene tantas columnas como el número de registros personalizados que se van a insertar.

En el caso de que la tabla contenga campos relacionados, se genera una matriz donde se almacenan los valores de los campos con los que se relaciona, de modo que durante el proceso de inserción se accede a esta matriz y se selecciona aleatoriamente una columna, para poder asignar valores que realmente existen a campos relacionados y no incurrir en errores debidos a la integridad referencial que se exige.

El motor comienza a construir una consulta *'Insert'* para cada tabla, en función de las características de los campos (relaciones, tipo, longitud, sistema gestor). Cuando finaliza su construcción, queda de la forma siguiente:

'INSERT INTO <*tabla*> (<*campo*$_1$>, <*campo*$_2$>, … , <*campo*$_n$>)
VALUES (<*valor*$_1$>, <*valor*$_2$>, … , <*valor*$_n$>)'.

- Si el campo está relacionado, el motor accede a la matriz en la que se han almacenado los valores de los campos con que se relaciona y aleatoriamente selecciona un valor contenido en esa tabla. Si se tratase de un campo con valor personalizado y ese valor no se encuentra en la matriz que contiene los valores de los campos con que se relaciona, se procede a la actualización de uno de los registros con que se relaciona, para la inclusión de ese valor.

- Si el campo no está relacionado con ningún otro, el motor genera aleatoriamente un valor para ese campo. Para ello analiza el tipo, el tamaño y el sistema gestor de base de datos sobre la que trabaja.

- En caso de que el tipo sea autoincremental se modifica la sentencia *'Insert'* de modo que no aparezca ni el nombre del campo ni su valor, dejando así que sea el propio sistema gestor el que administre los valores de ese campo.

- Una vez que se ha terminado de generar la consulta *'Insert'*, ésta se ejecuta sobre la base de datos y en caso de que se produzca algún error (normalmente debido a una clave principal repetida), se vuelve a construir una nueva consulta *'Insert'* y se vuelve a ejecutar. Si se produce más de un cierto número de errores consecutivos, el motor de inserción se detiene e informa al usuario de que se han producido un número elevado de errores, de este modo se evita que en caso de un error general, el motor quede atrapado en un bucle infinito y por lo tanto bloqueado.

En la figura 4.3.12 se observa el organigrama básico del motor de inserción.

5.2.6 Descripción del motor de toma de medidas

Se trata del núcleo de la herramienta y se encarga de controlar el motor de inserción de registros y medir los tiempos de respuesta de las bases de datos a las consultas realizadas por ésta, ya sean personalizadas o predefinidas.

Las tareas básicas que realiza son:

- *Lectura de la configuración*: Procede a la lectura de los diversos parámetros de configuración que ha realizado el usuario.
- *Construcción del archivo de resultados*: Contiene todos los datos recogidos durante la medición (base de datos, fecha, versión del programa, nombre del *driver* empleado para la gestión de la base de datos, versión del *driver*, número de repeticiones de la toma de medida, número de registros, vista SQL de la consulta, tiempo que tarda en realizarse la consulta).
- *Lanzamiento del motor de inserción de registros*: Crea un hilo de baja prioridad para evitar bloqueos o mal funcionamiento del ordenador, y lanza el motor de inserción de registros administrándole información sobre las tablas que tiene que crear.
- *Análisis previo de la consulta a realizar* y toma de tiempos y almacenamiento de éstos: Identifica el tipo de consulta que se va a realizar, caso de que el usuario desee visualizar los resultados de las consultas, almacena las salidas que producen los sistemas gestores de bases de datos y almacena el tiempo que tarda el sistema gestor en realizar la consulta.

    - *SELECT*: Se transmite al sistema gestor de base de datos la consulta exactamente igual a como ha sido construida por el usuario o como ha sido definida previamente y se recogen los datos aportados por el sistema gestor.
    - *INSERT*: Se analiza completamente la instrucción y se construye una instrucción de borrado que restaura el estado de las tablas al que tenían antes de haberse ejecutado la instrucción *'Insert'*.

- *DELETE*: Se analiza la consulta extrayendo el nombre de la tabla afectada y se crea un árbol con el nombre de todas las tablas relacionas con ésta. Se realizan copias de seguridad de las tablas afectadas (adecuándolas al sistema gestor sobre el que se trabaja). Si van a ser borradas utilizando el borrado en cascada propio del sistema gestor (el *driver* ODBC para Microsoft Access no permite borrado en cascada) se transmite la instrucción al sistema gestor y éste se encarga de borrar todas las tablas implicadas. En caso de que se deseen borrar registros utilizando la emulación que proporciona la herramienta, se generan instrucciones de borrado analizando las relaciones y los registros afectados y se transmiten al sistema gestor de modo que no se incurra en errores por violaciones de la integridad referencial. Tras la toma de medidas se restauran las tablas afectadas incluyendo claves e índices, para lo cual se utilizan los datos obtenidos previamente del análisis de las tablas.
- *UPDATE*: La respuesta del programa ante una instrucción de este tipo va a ser la misma que ante una instrucción *'delete'*, con la salvedad de que en lugar de borrado de registros se realiza una actualización de éstos.

En la figura 4.3.14 se observa el diagrama de flujo básico del motor de toma de medidas.

6. Pruebas experimentales

6.1 Descripción del banco de pruebas.

Todas las pruebas y medidas han sido tomadas en un laboratorio que cuenta con veinte ordenadores conectados en LAN.

Debido a que las pruebas realizadas tienen como principal objetivo la medición del rendimiento obtenido por los diferentes tipos de mapeos de objetos sobre bases de datos relacionales así como el rendimiento reportado por los sistemas gestores de bases de datos empleados resulta indispensable la descripción tanto de los equipos empleados como el software empleado, ya sean sistemas operativos, drivers o versiones de los sistemas gestores de bases de datos.

Equipos empleados.

A continuación se describen las características de los equipos empleados tanto a nivel de red como de computador.

- Fujitsu Siemens Scenic

Procesador	
Marca	Intel
Modelo	Northwood HyperThreading
Frecuencia de funcionamiento	3000 MHz
Cache L2	512 KB
Placa Base	
Marca	Fujitsu Siemens
Modelo	D1562
Chipset	Intel Springdale-G i865G
Reloj Bus Principal	800 MHz
Memoria	
Marca	Infineon Technologies AG

Tipo	DDR SDRAM
Tamaño del módulo	512 MB
Número de módulos	2
Total disponible	1024 MB
Ancho del módulo	64 bits
Dual Channel	Activado
Ancho del Bus	128 bits
Reloj real	200 MHz
Reloj efectivo	400MHz
Latencias CL-RCD-RP-RAS	3.0-3-3-8
Disco Duro	
Marca	Samsung
Modelo	SP0802N, SpinPoint P80
Capacidad	80 GB
Velocidad de rotación	7200 RPM
Tamaño del buffer	2 MB
Modo de transferencia activa	UDMA 5
Coeficiente máximo de transferencia interna	741 Mbit/s
Tiempo de busqueda medio	8.9 ms
Tarjeta de Red	
Marca	Intel
Modelo	PRO/1000 CT 82547 EI
Tipo	Integrada en placa base
Máxima velocidad	1 Gbit/s
Velocidad conexión actual	100 Mbit/s

- Switch

Fabricante	Cisco
Modelo	2950G-48
Número de puertos	48 de 100 Mbit/s y 2 de 1 Gbit/s
Ancho de banda máximo	13.6 Gbit/s
Máxima velocidad de direccionamiento	10.1 Mpps

Especificaciones del software.

A continuación se describe el software empleado.

Sistema Operativo	
Fabricante	Microsoft Corporation
Sistema	Microsoft Windows XP Pro
Versión	2002
Service Pack	1
Microsoft Access	
Fabricante	Microsoft Corporation
Versión	2003 (11.5614.5606)
Motor Jet	4.0
Nombre del Driver	JDBC-ODBC Bridge (odbcjt32.dll)
Versión del Driver	2.0001 (04.00.6019)
Oracle	
Fabricante	Oracle
Versión	9i (9.2.0.1.0)
Sistema Operativo	Windows XP Pro 2002 SP 1
Nombre del Driver	Oracle JDBC Driver
Versión del Driver	10.1.0.4.0
MySQL	
Fabricante	MySQL AB
Versión	4.1.11
Sistema Operativo	Windows XP Pro 2002 SP1
Nombre del Driver	MySQL-AB JDBC Driver
Versión del Driver	3.1.8 Revisión 1.27.4.64
PostgreSQL	
Fabricante	Comunidad PostgreSQL*
Versión	7.4.1
Sistema Operativo	Linux Mandrake 10 2.6.3-7
Nombre del Driver	PostgreSQL Native Driver
Versión del Driver	PosgreSQL 8.0 JDBC3 con SSL Built 311

Topología de red

Se trata de una Gigabit Ethernet funcionando a 100 Mbit/s limitada debido a la velocidad máxima del switch empleado (Cisco Catalyst 2950G-48 con velocidad del puerto de 100 Mbit/s).
En la figura 6.1 podemos visualizar la topología de la red.

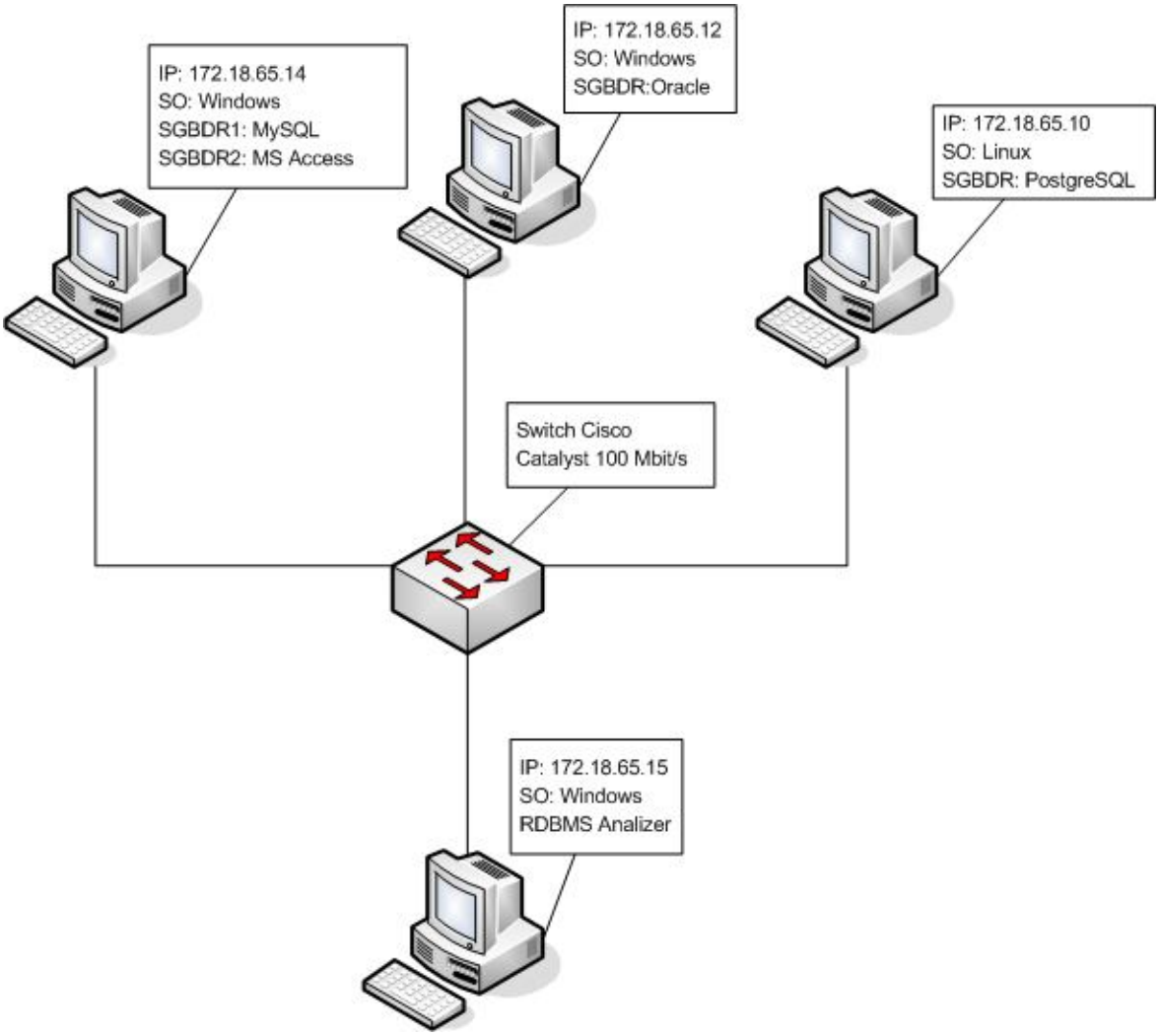


Figura 6.1: Topología de red.

6.2 Pruebas realizadas

Dada la enorme cantidad de pruebas realizadas, toda la información se traslada al anexo 1.

7. Resultados experimentales

Lanzamos todas las baterías de pruebas automatizadas predefinidas en los ficheros de texto sobre los diferentes SGBDR y procedemos a visualizar y analizar los resultados, utilizando el módulo de visualización y análisis que incorpora la propia herramienta.

Vamos a realizar el análisis en dos partes, una se va a centrar en el rendimiento de los diferentes SGBDR empleando diferentes patrones de mapeo y la otra parte se va a centrar en las diferencias de rendimiento de los patrones de mapeo sobre un único SGBDR. Dada la enorme cantidad de pruebas que se han realizado, se va a presentar únicamente los resultados más representativos.

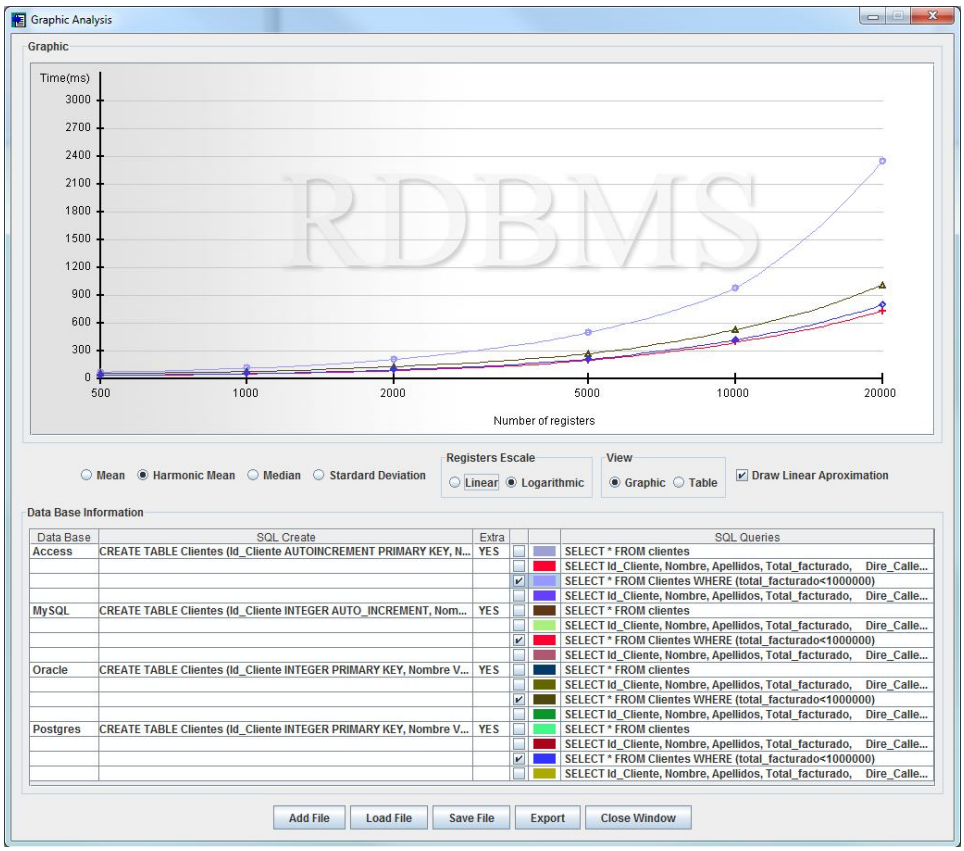
7.1 Rendimiento de los diferentes SGBDR.

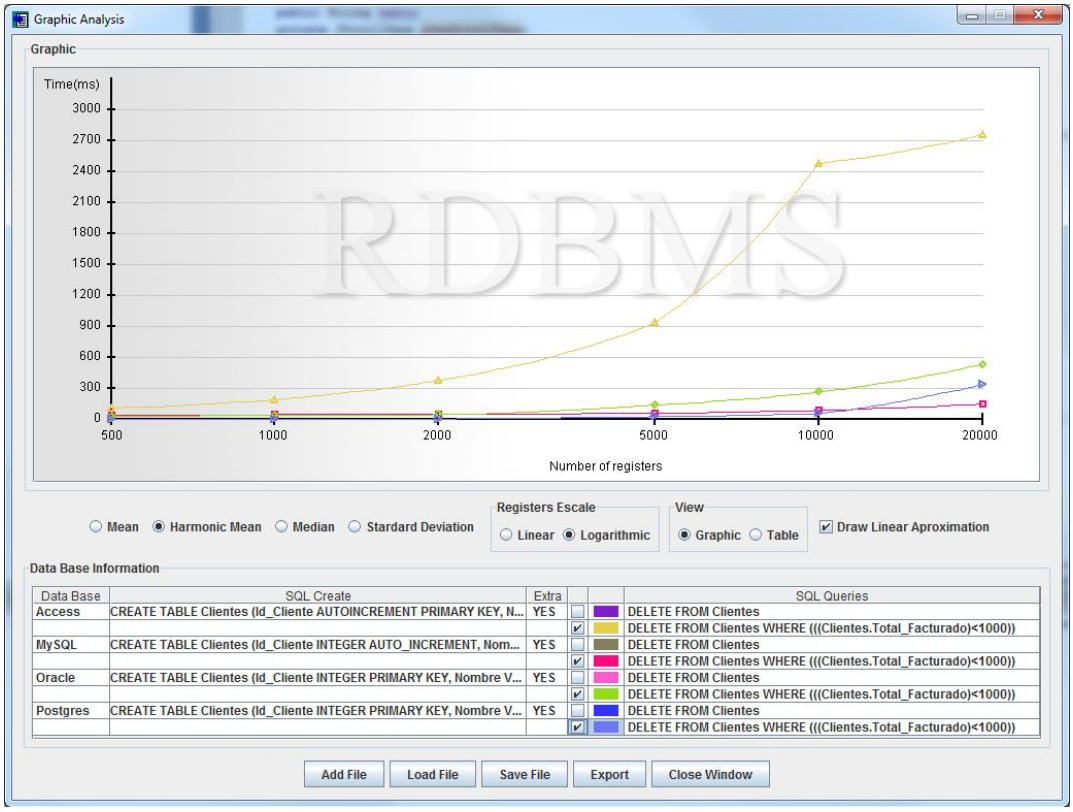
Nos vamos a centrar en analizar los resultados de rendimiento obtenidos al emplear diferentes patrones de mapeo sobre diferentes SGDBR, y tras el análisis vamos a extraer una serie de conclusiones respecto al comportamiento de los diferentes SGDBR.

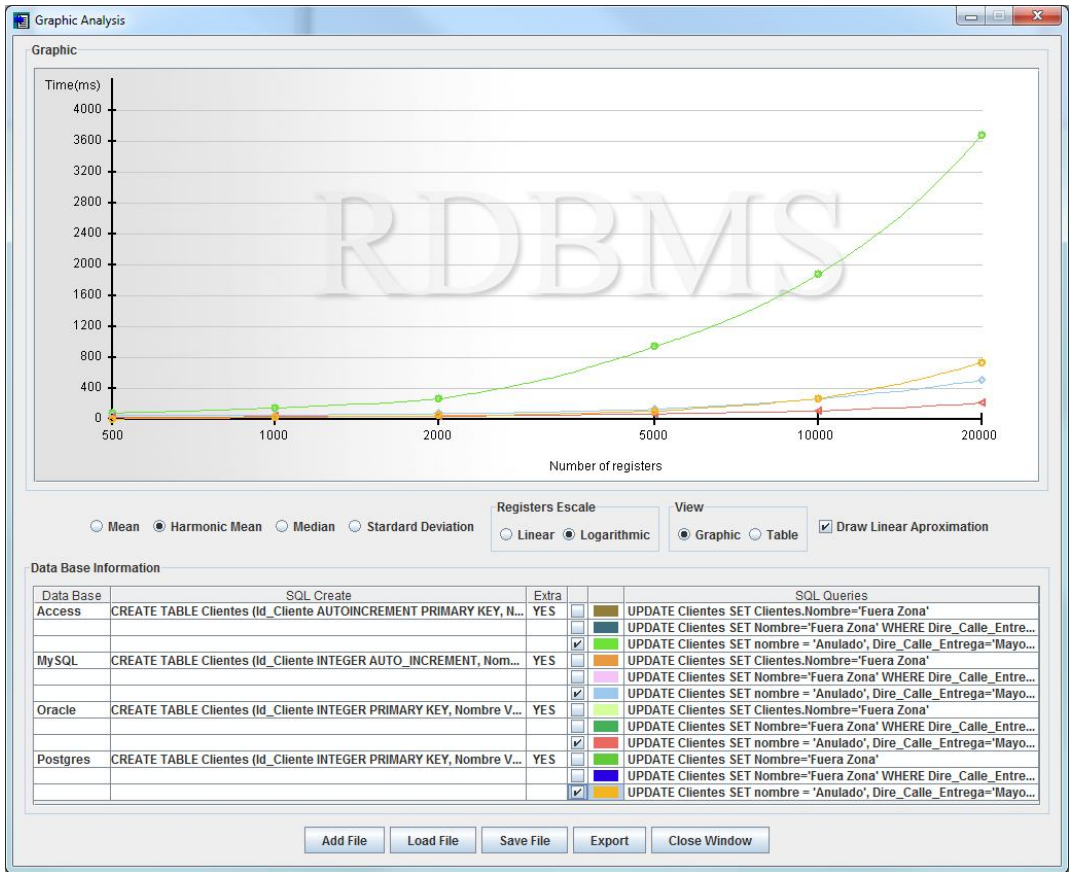
En primer lugar presentamos las gráficas resultantes de aplicar los diferentes tipos de mapeo de objetos a los diferentes SGBDR.

7.1.1 Agregación.

- Tabla sencilla

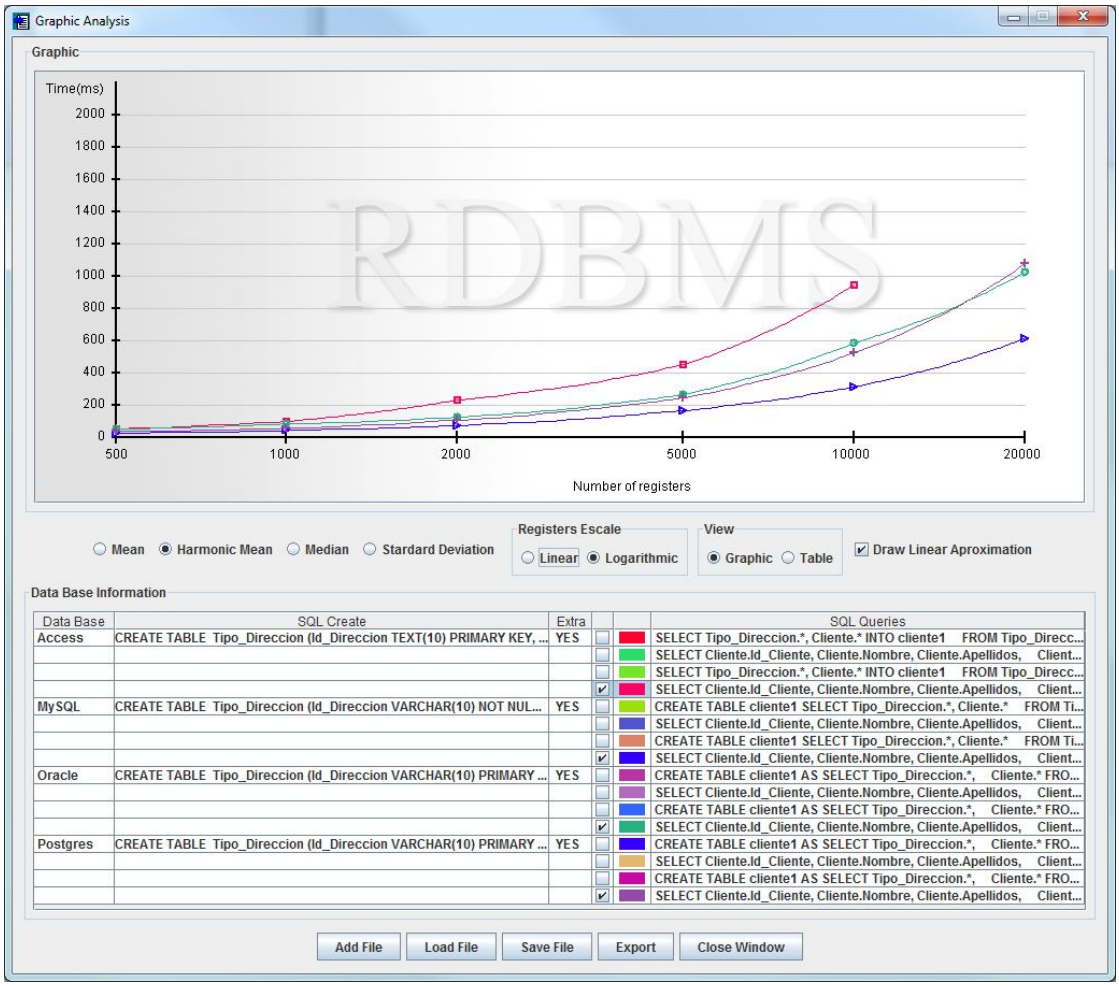


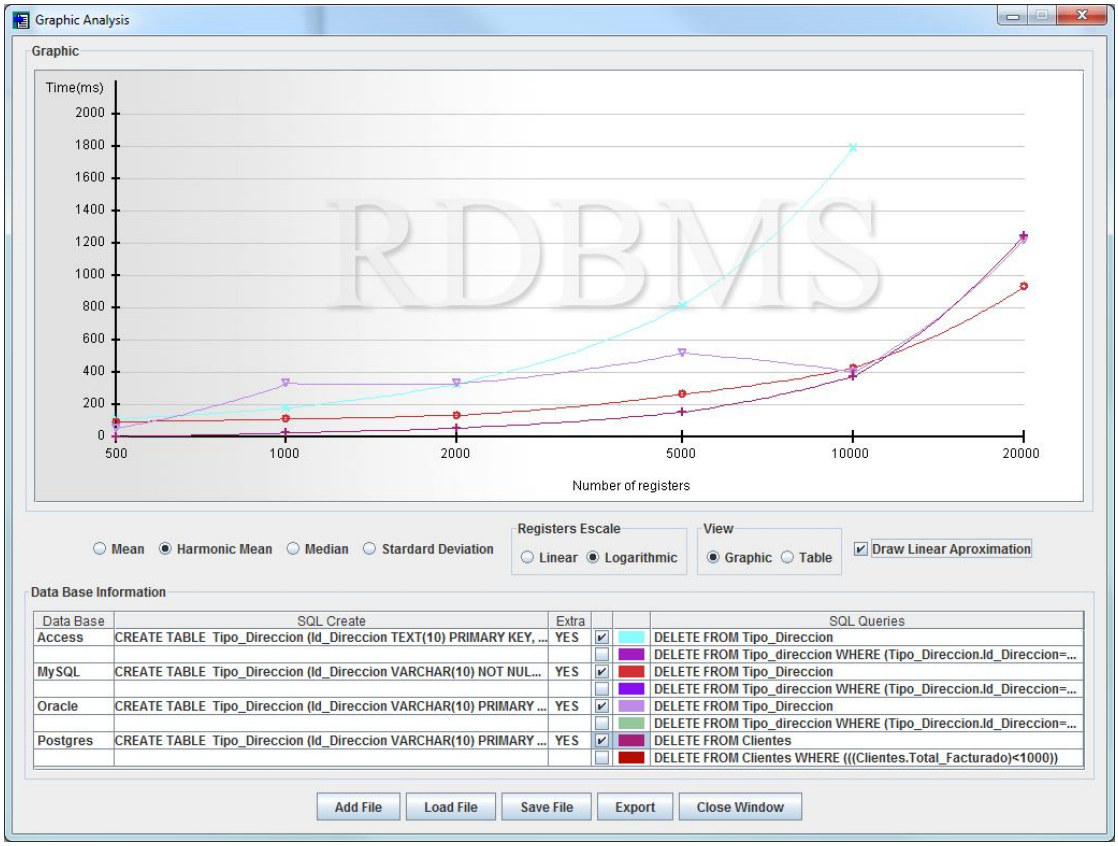


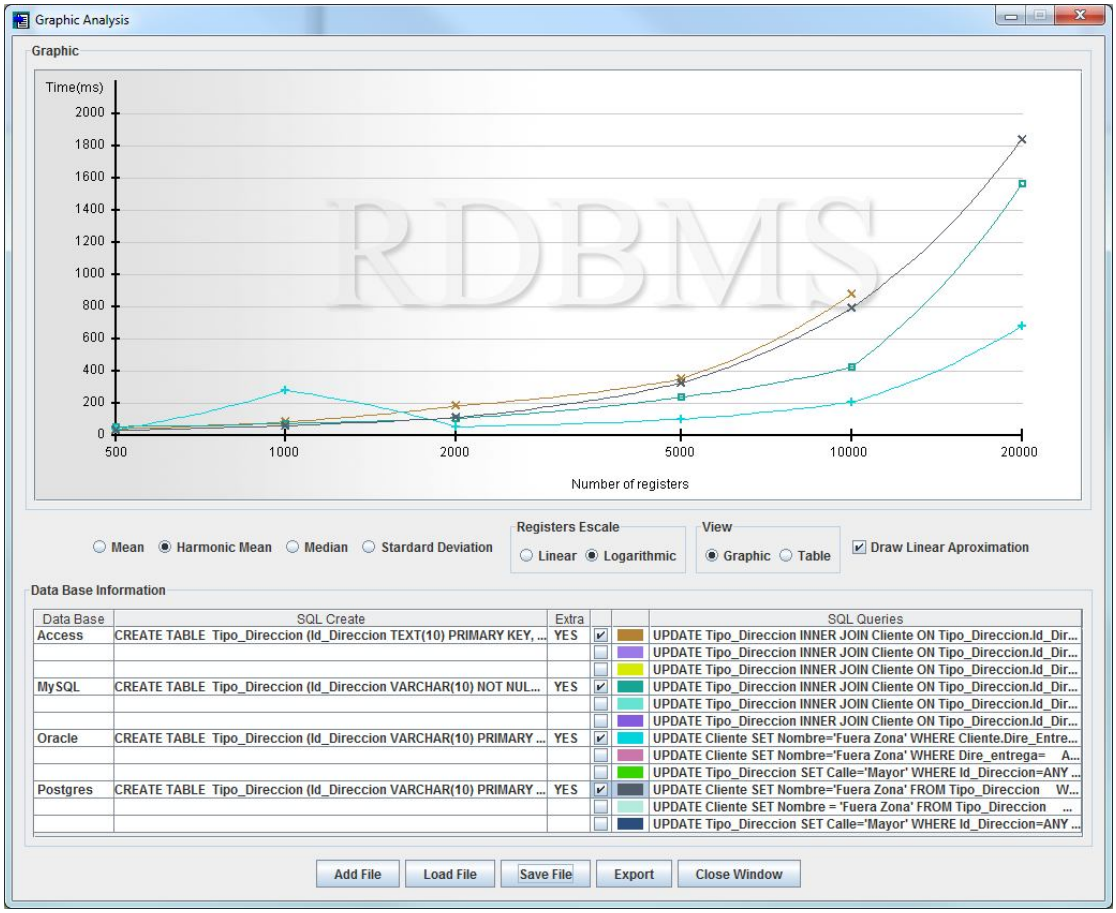


En las gráficas anteriores podemos observar como Access tiene un rendimiento muy inferior al resto de SGBDR, respecto a los demás SGBDR, pese a que en líneas generales obtienen unos resultados con un comportamiento muy similar, es MySQL el que nos proporciona un mayor rendimiento en peticiones de consulta, en el caso de peticiones de borrado se encuentra muy igualado con Postgresql e incluso en algunas peticiones de borrado es Postgresql el que ofrece un mayor rendimiento, en el caso de peticiones de actualización de registros es Oracle el que ofrece el mejor rendimiento.

- Clave extranjera



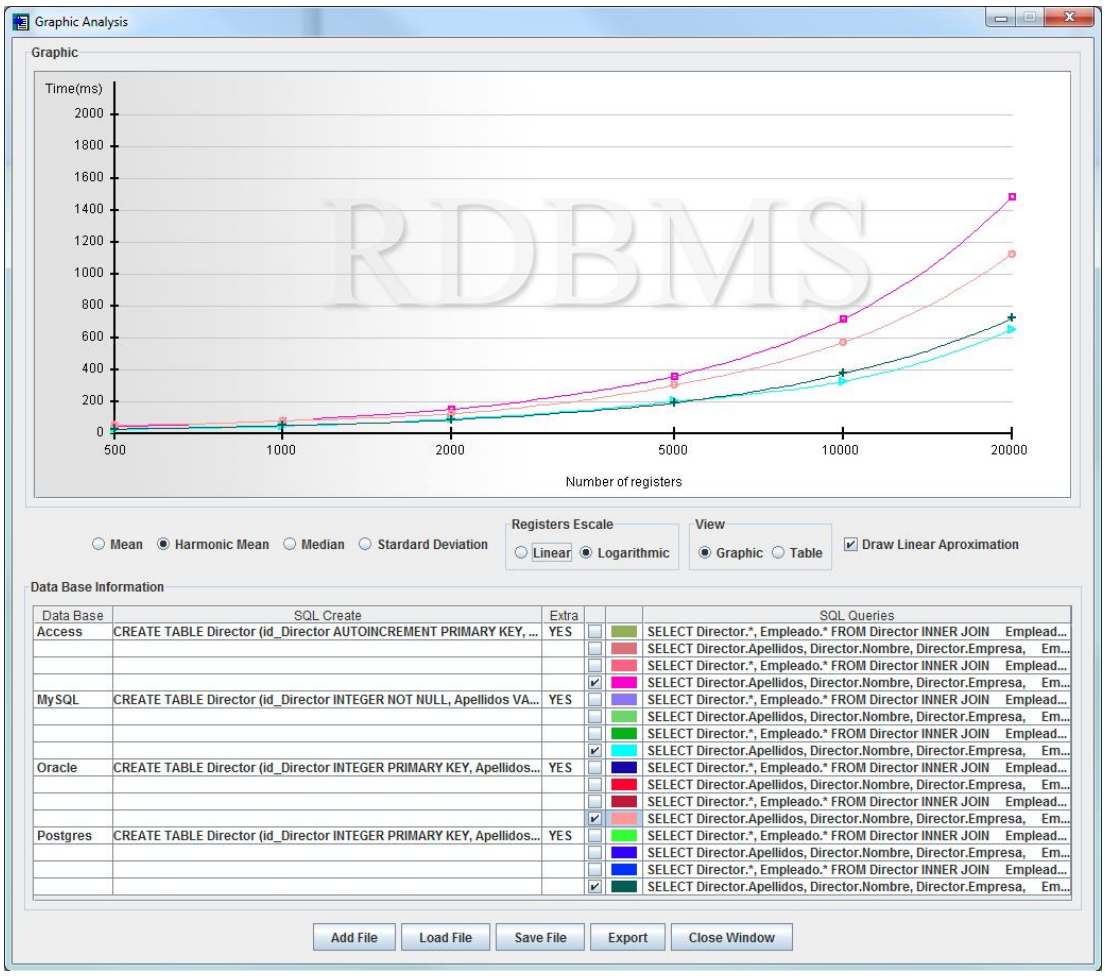


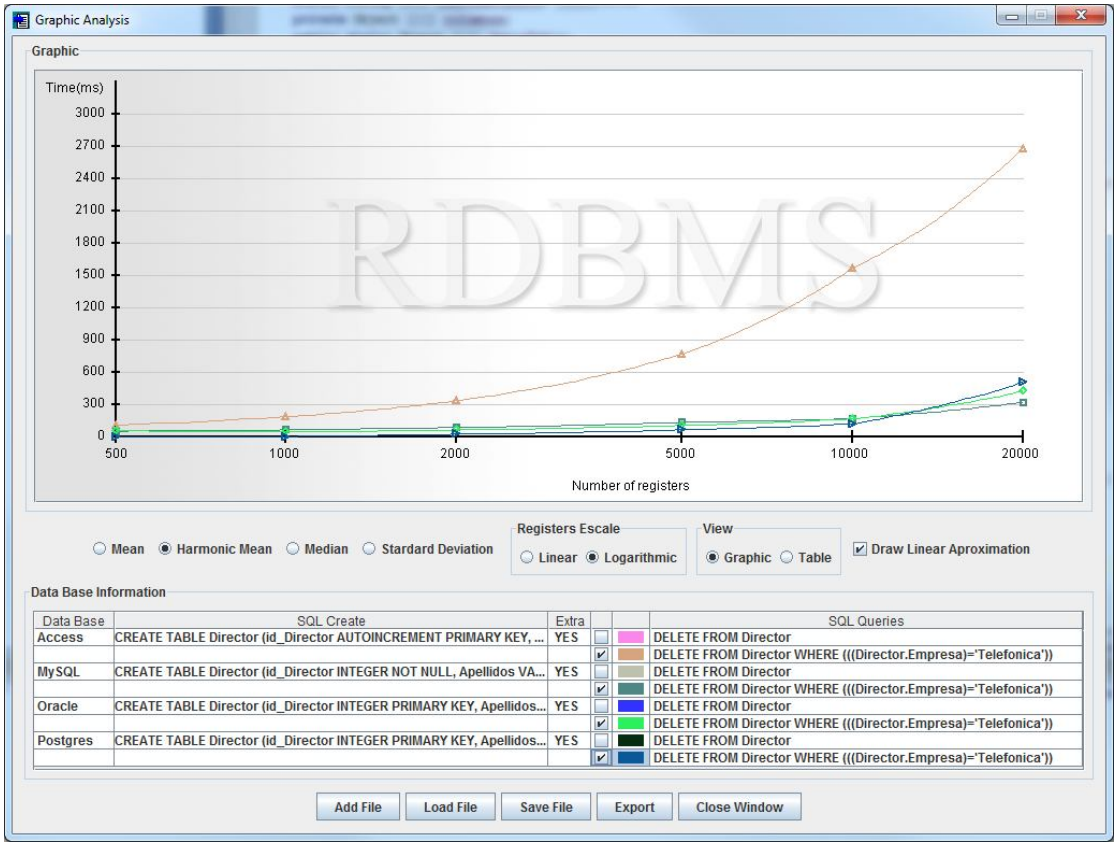


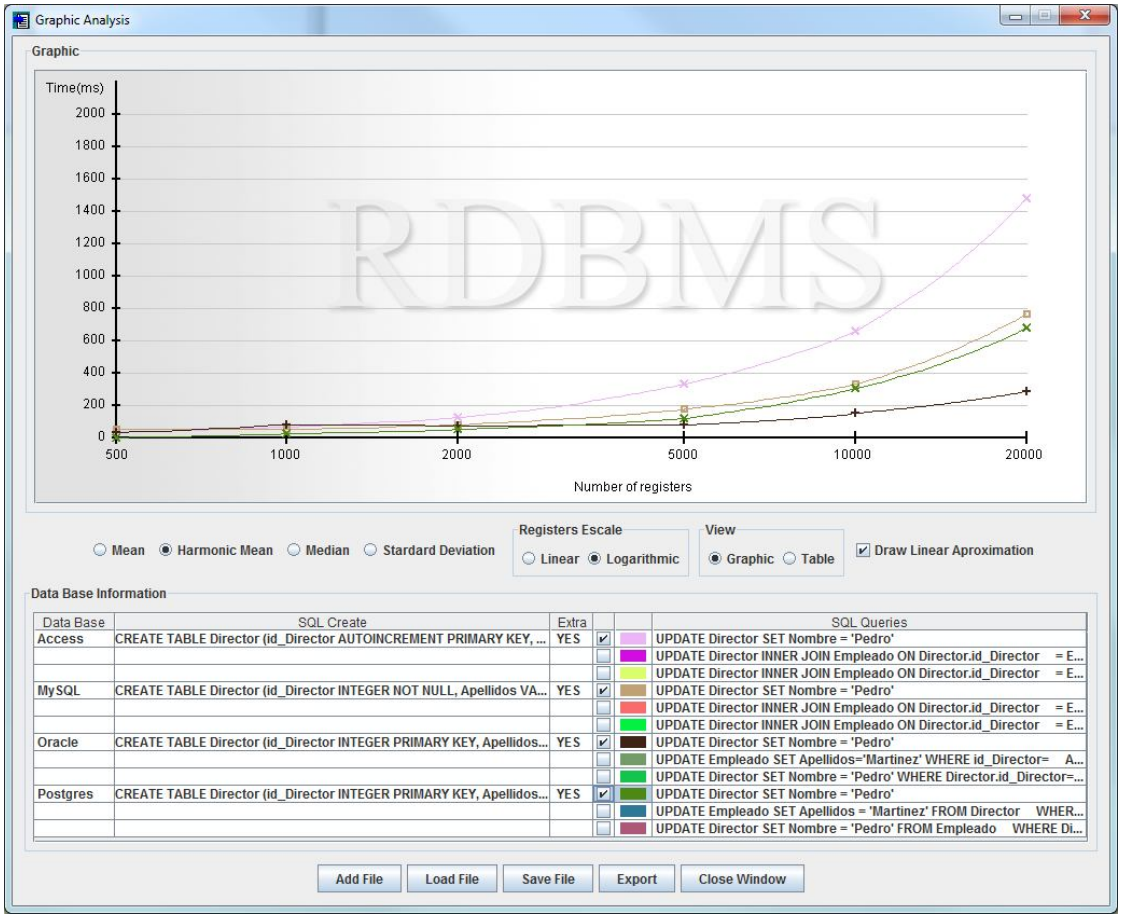
En el caso de clave extranjera las conclusiones obtenidas son las mismas que para el caso de tabla sencilla. Access ofrece el peor rendimiento y MySQL ofrece en líneas generales el mejor rendimiento, salvo en las actualización donde vuelve a ser Oracle el que claramente obtiene el mejor rendimiento, aumentando las diferencias con el aumento del número de registros sobre los que se trabaja.

7.1.2 Asociación

- Clave extranjera

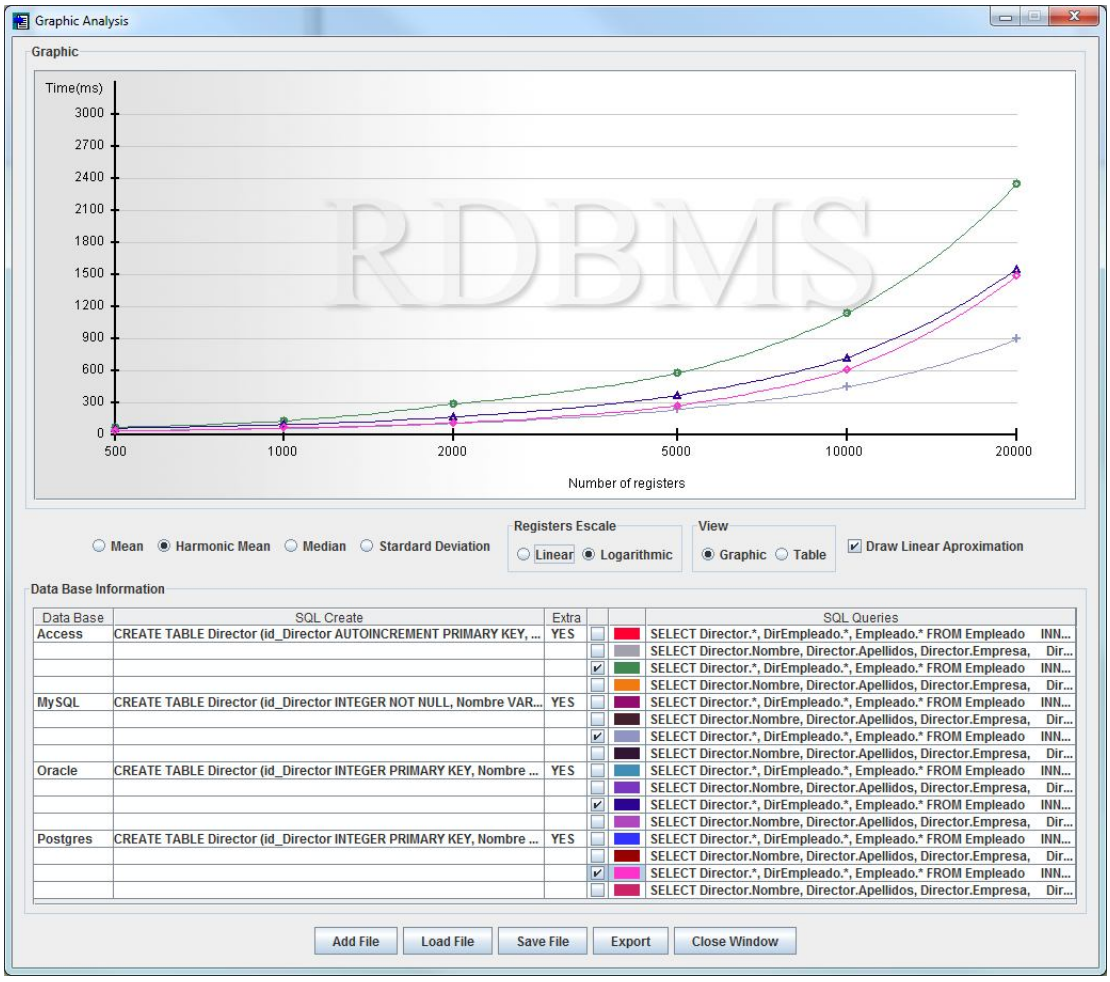


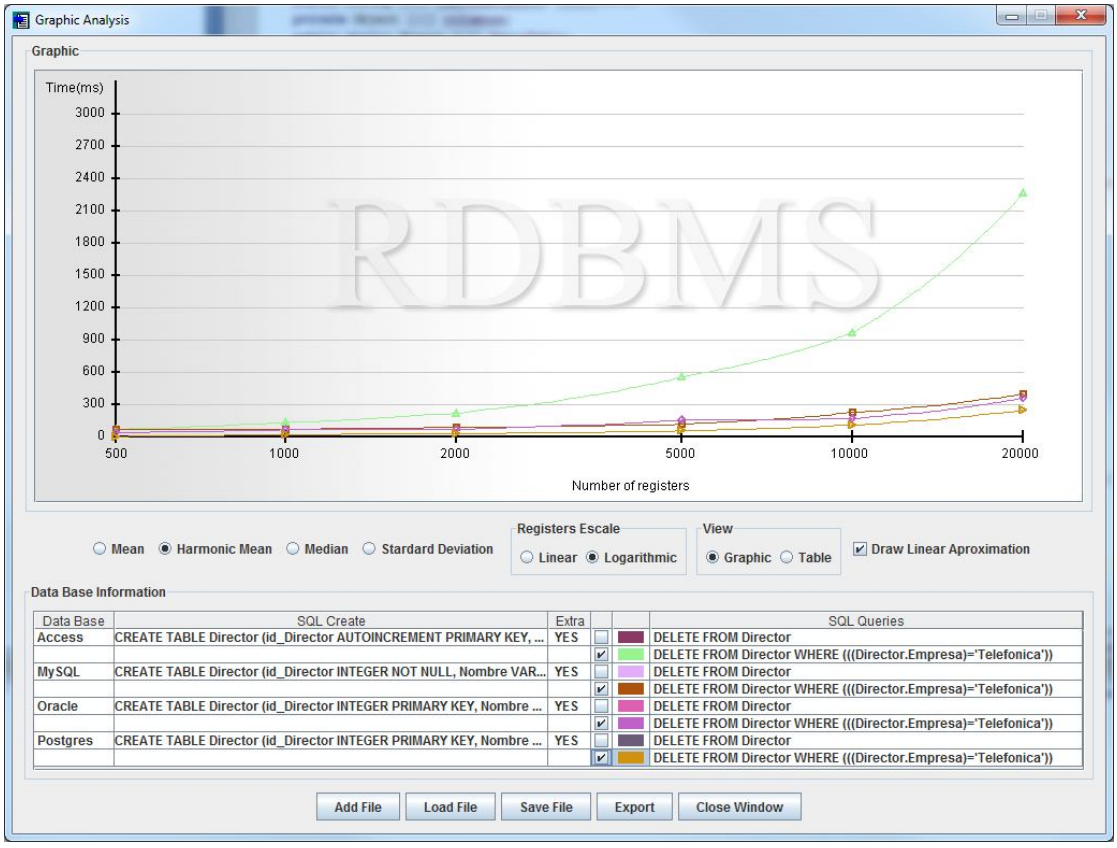


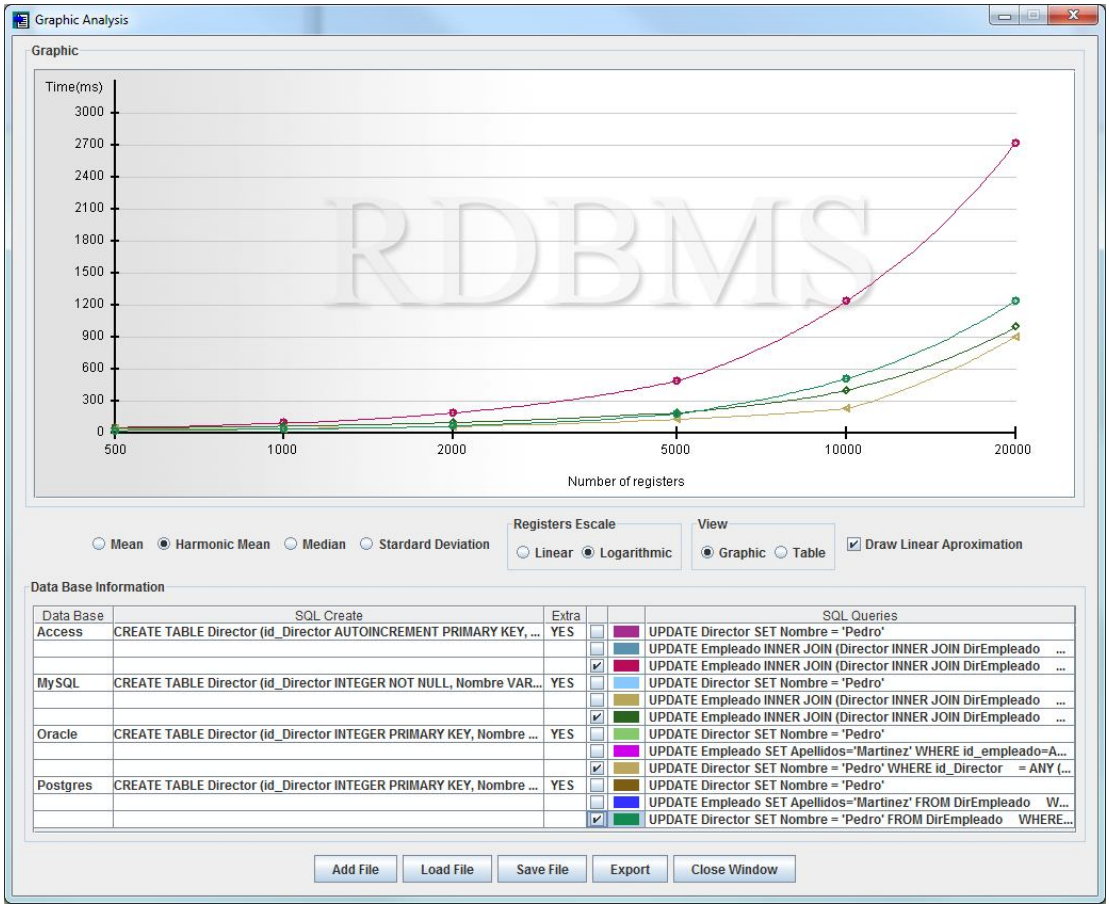


Access vuelve a ser el que ofrece un peor rendimiento. Respecto a peticiones de consulta MySQL sigue siendo el más rápido, en peticiones de actualización Oracle presenta el mejor rendimiento. En peticiones de borrado Postgresql se coloca en primer lugar.

- Tabla asociación.



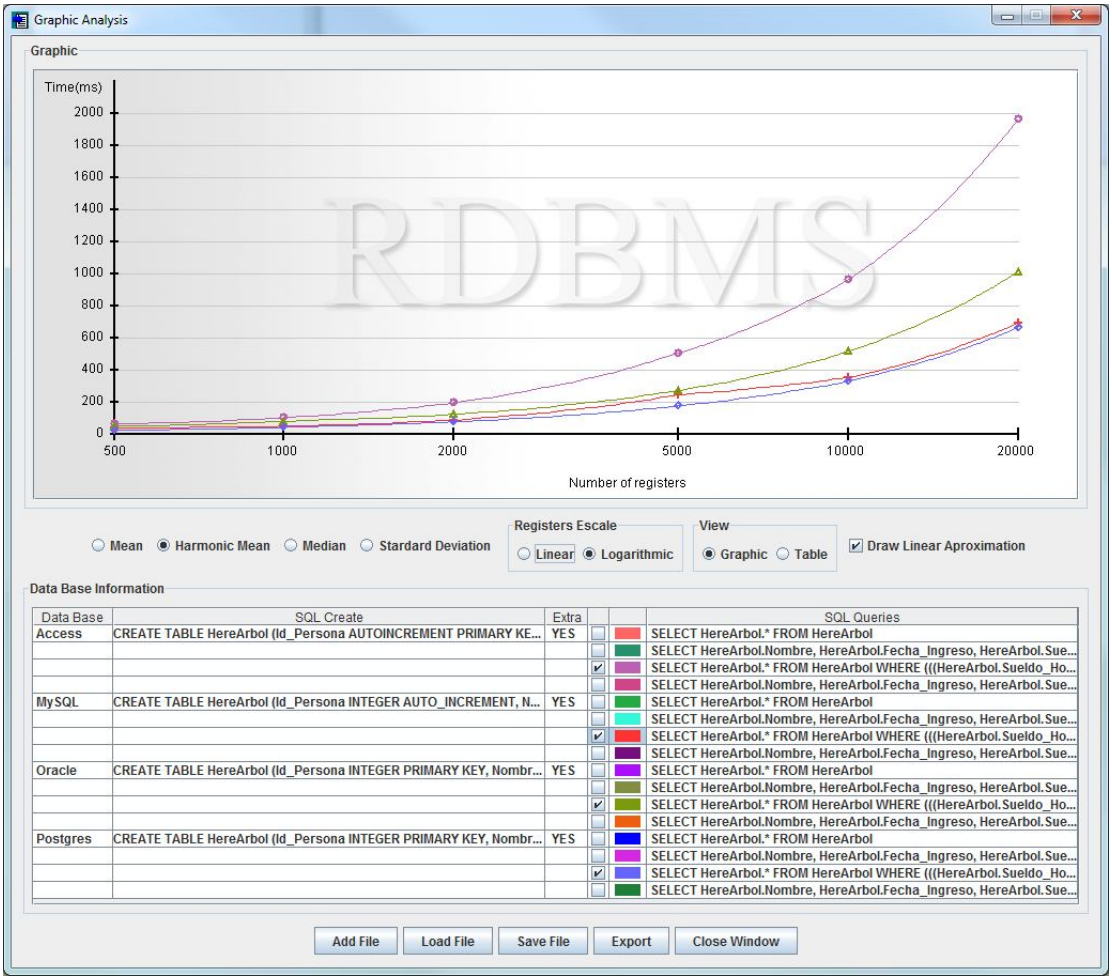


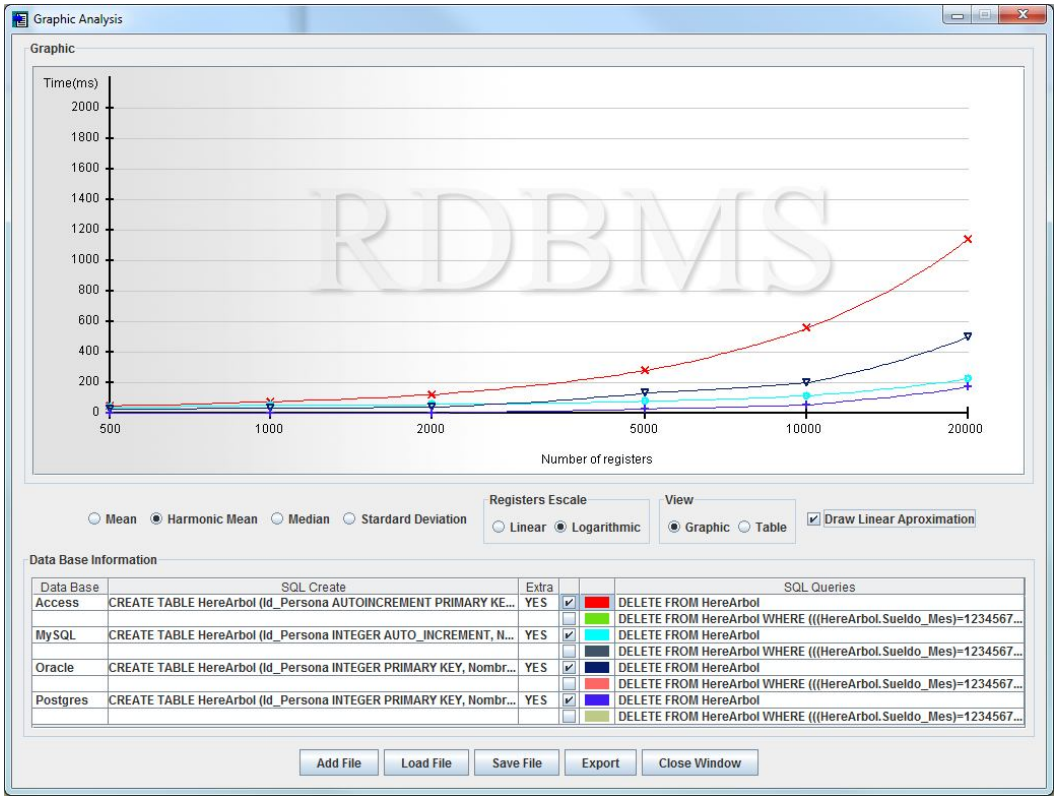


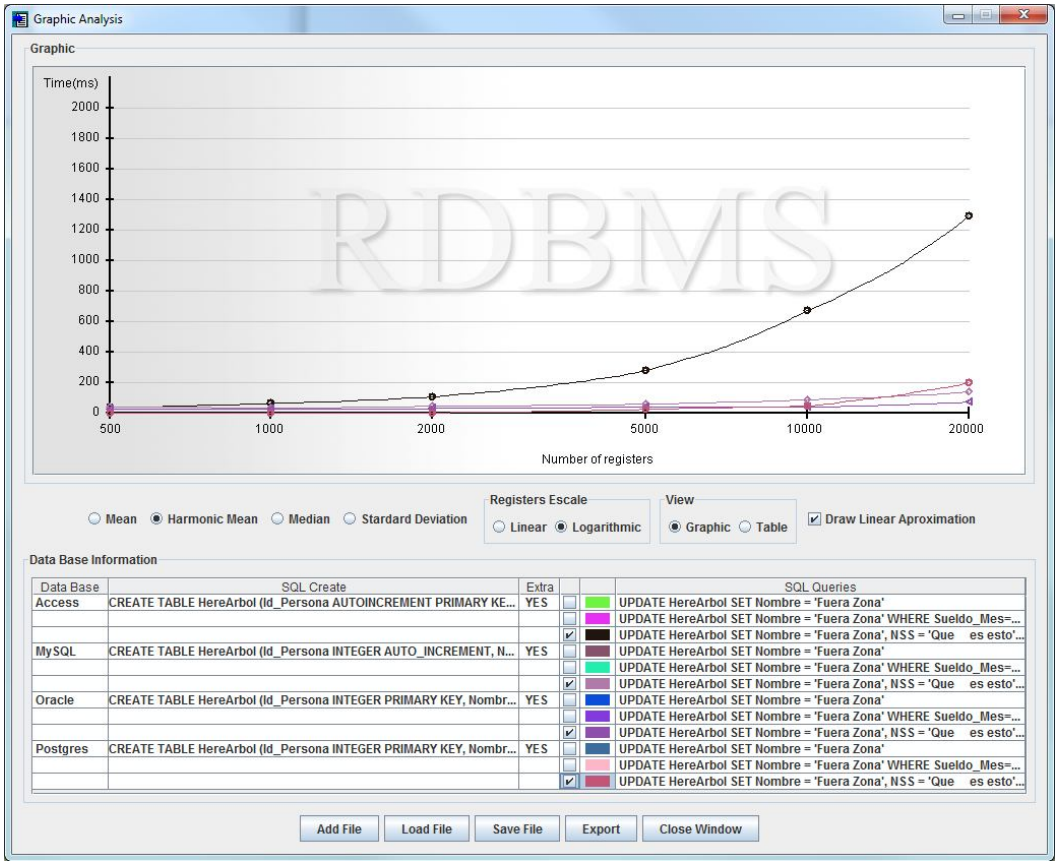
Las conclusiones para este mapeo asociación mediante tabla son exactamente las mismas que para los casos anteriores. Access el que ofrece un peor rendimiento en todas las pruebas realizadas, MySQL el mejor rendimiento en peticiones de consulta, Postgresql en peticiones de borrado y Oracle en peticiones de actualización.

7.1.3 Herencia.

- Árbol.

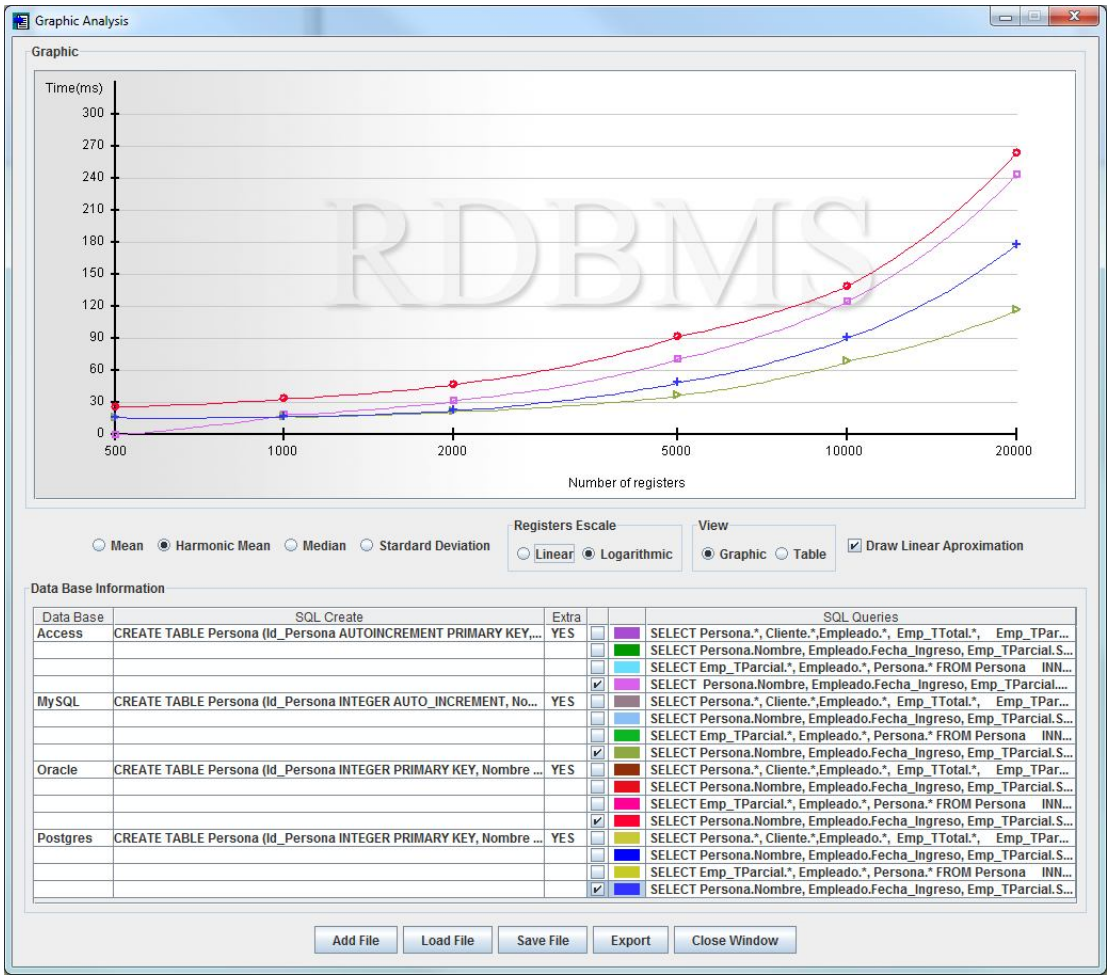


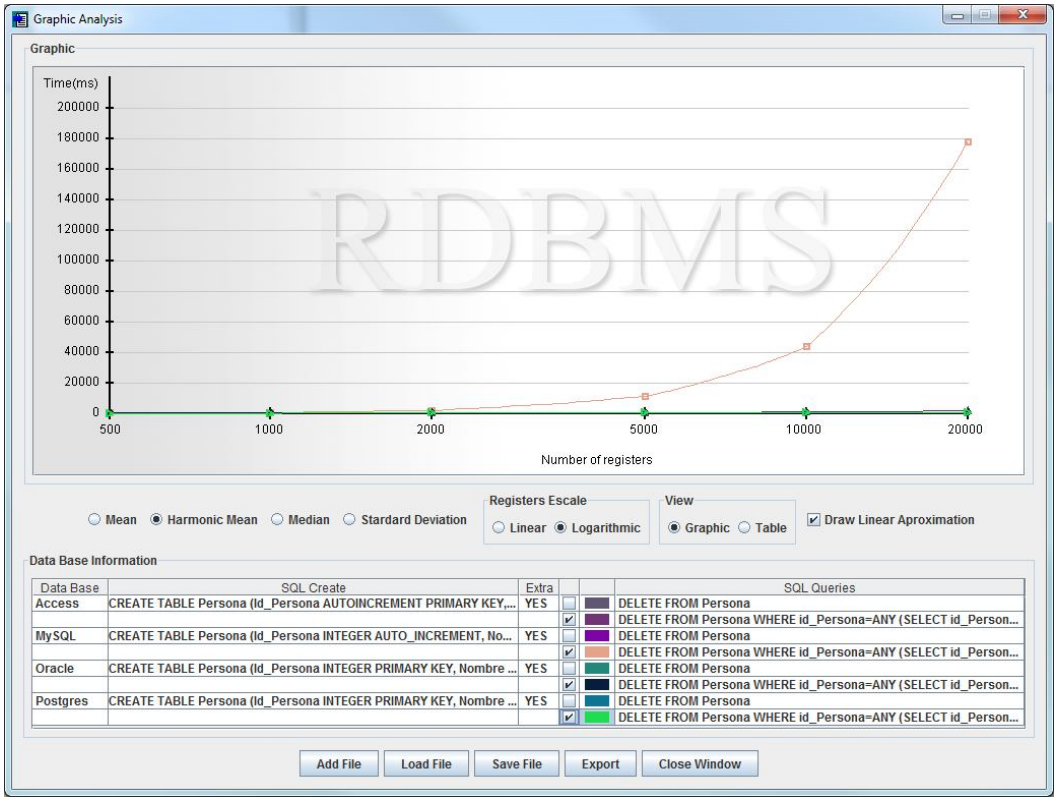


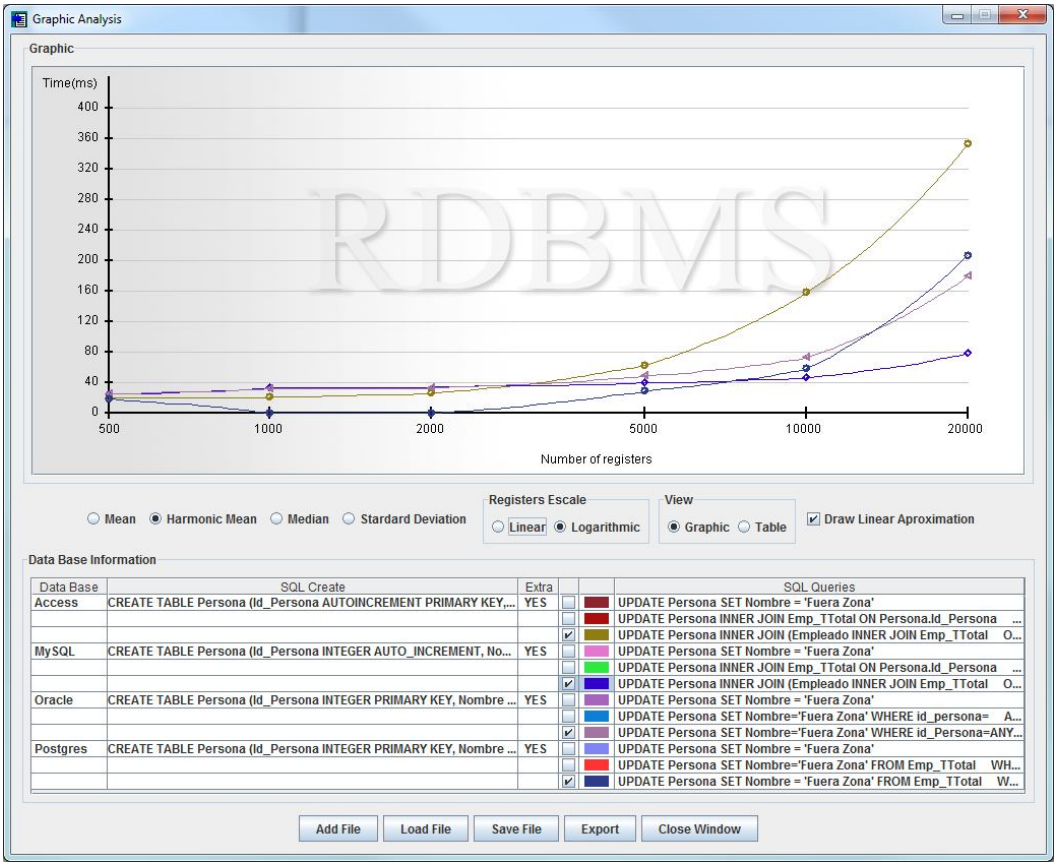


En las pruebas realizadas sobre el mapeo de la herencia utilizando el patrón de árbol una vez más Access obtiene con diferencia los peores resultados, Postgresql obtiene los mejores resultados en las peticiones de consulta, esta variación respecto a los casos anteriores puede ser debido a que el tamaño del registro a aumentado y en este caso Postgress los maneja de una manera más eficiente que MySQL, en las peticiones de borrado vuelve a ser el más rápido Postgresql y en las peticiones de actualización Oracle sigue siendo el más rápido.

- Una clase una tabla.

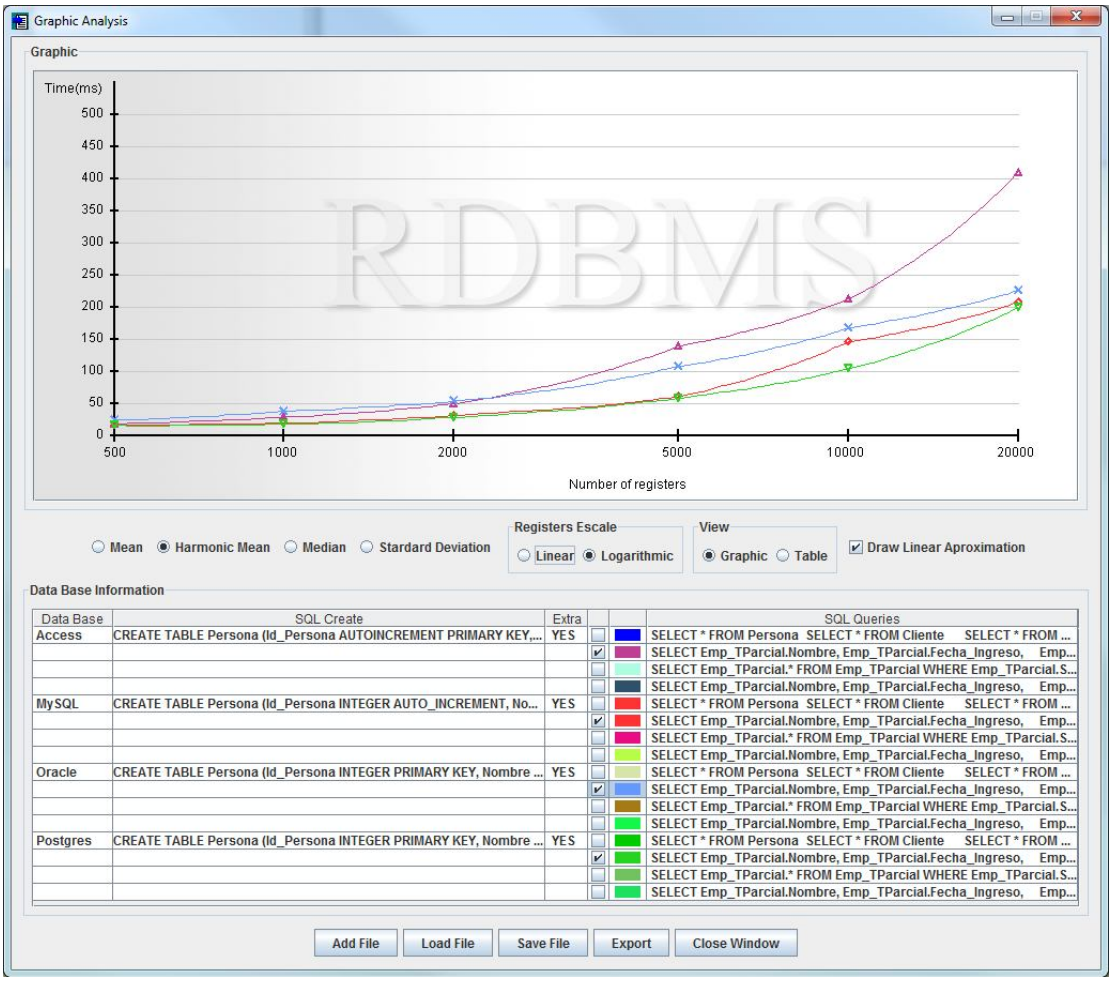


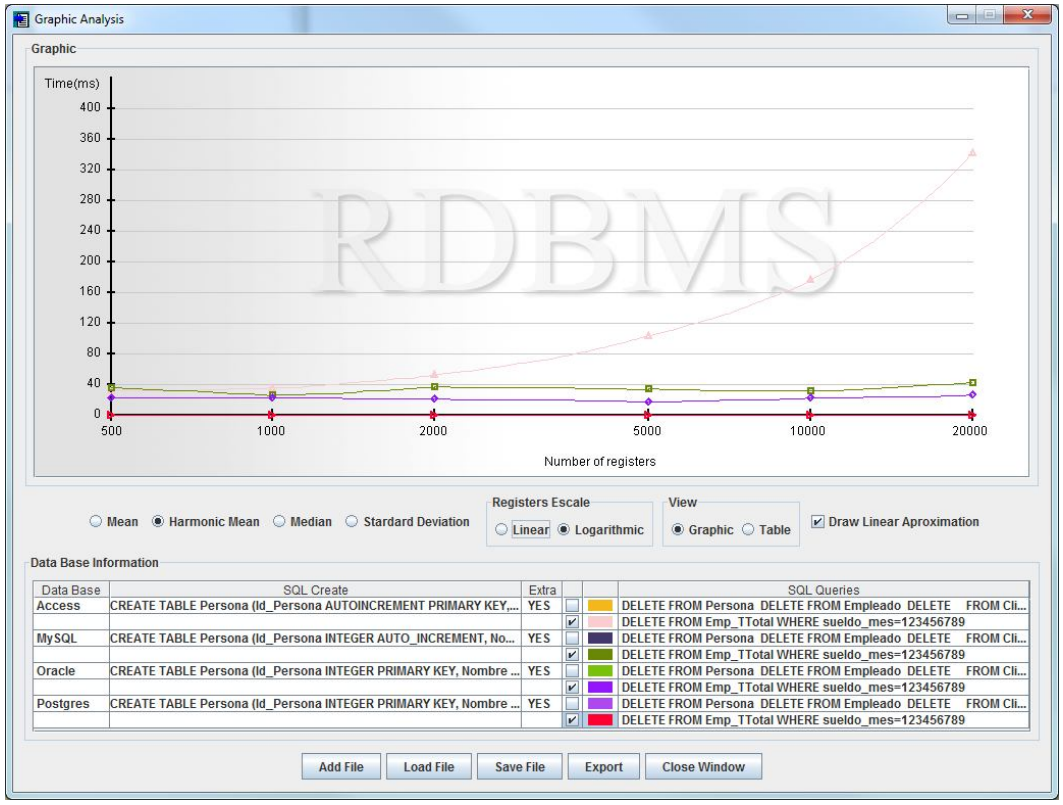


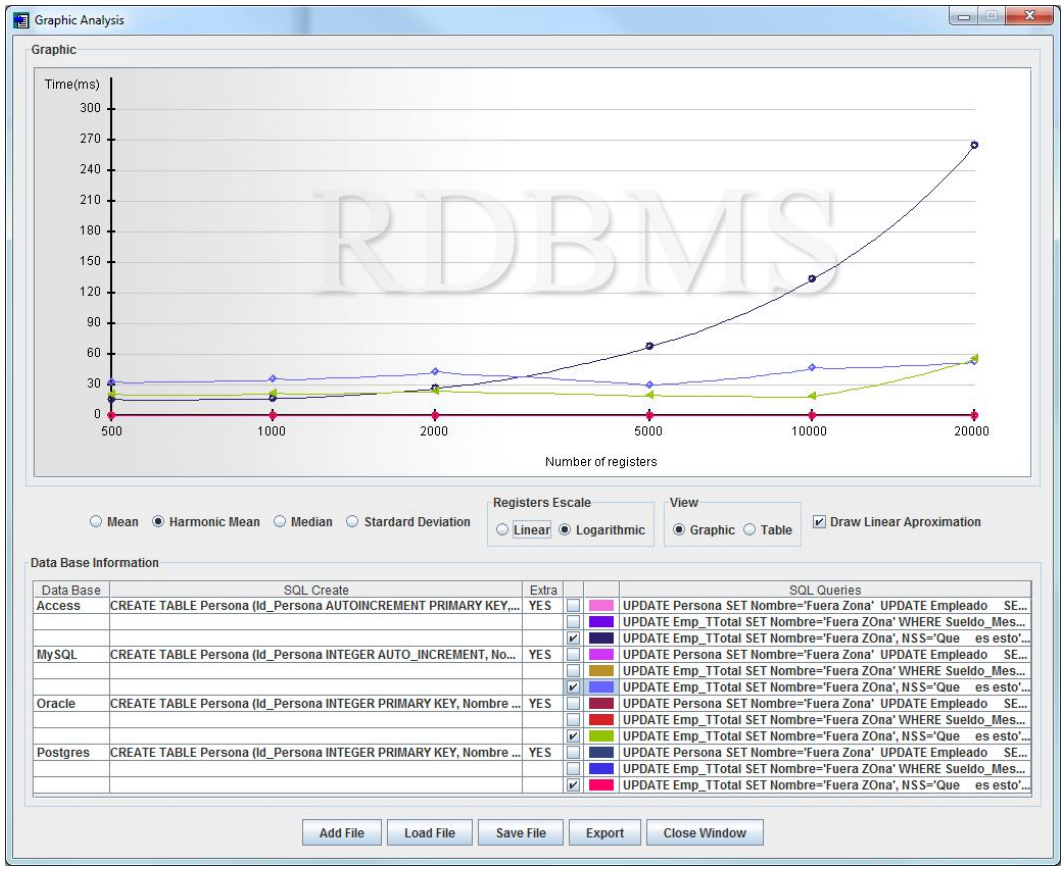


Access sigue presentando un rendimiento significativamente peor que el resto del los SGDBR, acentuado sobretodo en las peticiones de borrado, en la peticiones de consulta MySQL obtiene los mejores resultados, en las de borrado vuelve a ser el más rápido Postgresql y es llamativo ver que esta vez en las peticiones de actualización es MySQL el que presenta un mejor comportamiento afectándole en menor medida el aumento del número de registros, mientras que Postgresql ofrece el mejor rendimiento con un número bajo de registros va perdiendo la primera posición con el aumento del número de registros.

- Una ruta de herencia una tabla.







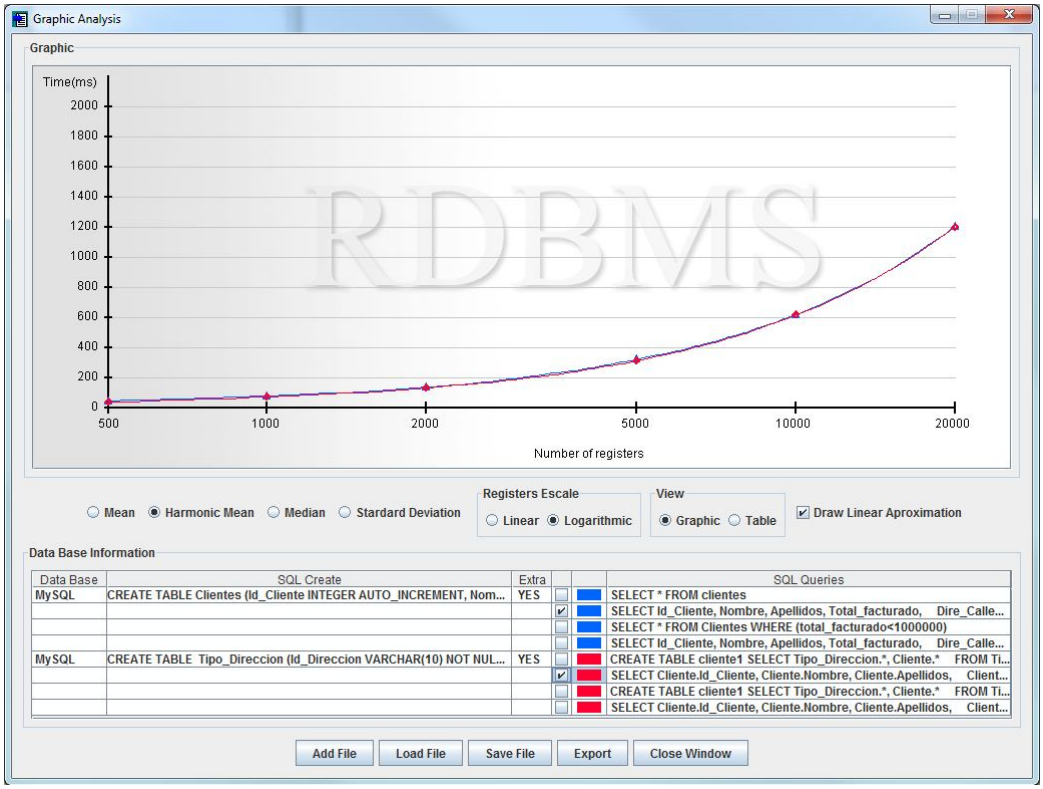
Access obtiene los peores resultados, en las peticiones de consulta el más rápido es Postgresql, pero seguido muy de cerca por MySQL. En las consultas de borrado y actualización el más rápido es Postgresql, aunque hay que destacar que salvo Access el resto de SGBDR también obtienen unos tiempos muy bajos e independientes del número de registros sobre el que trabajan.

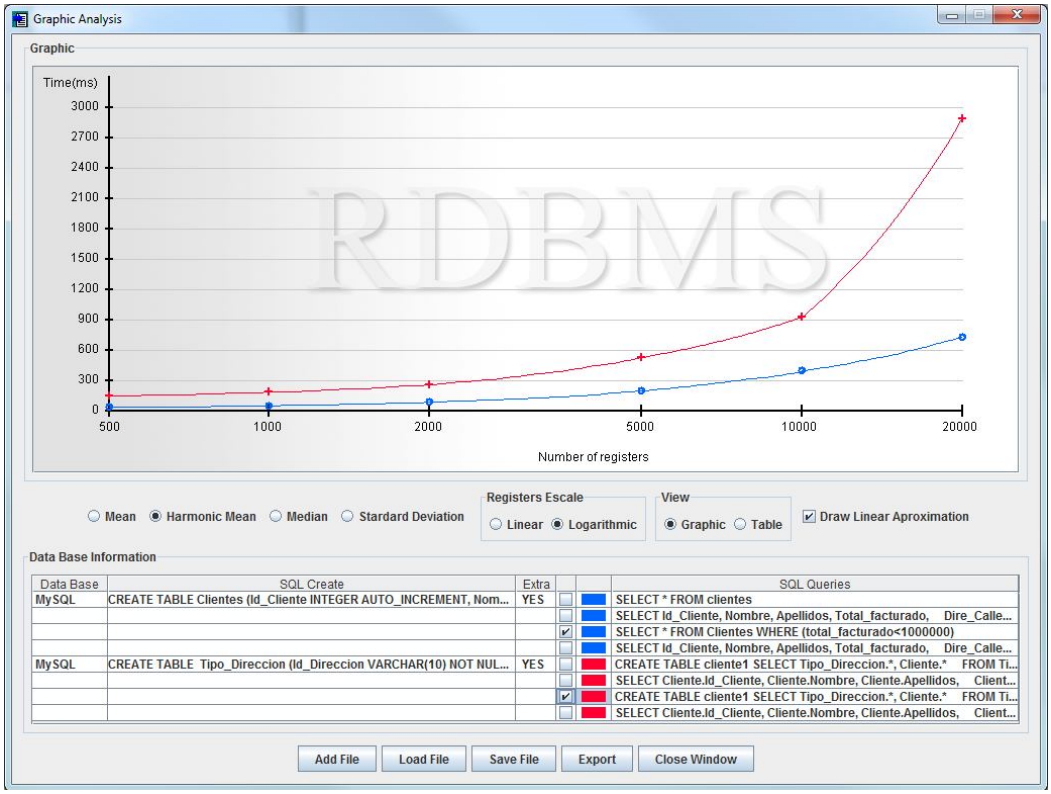
7.2 Rendimiento de los diferentes patrones de mapeo.

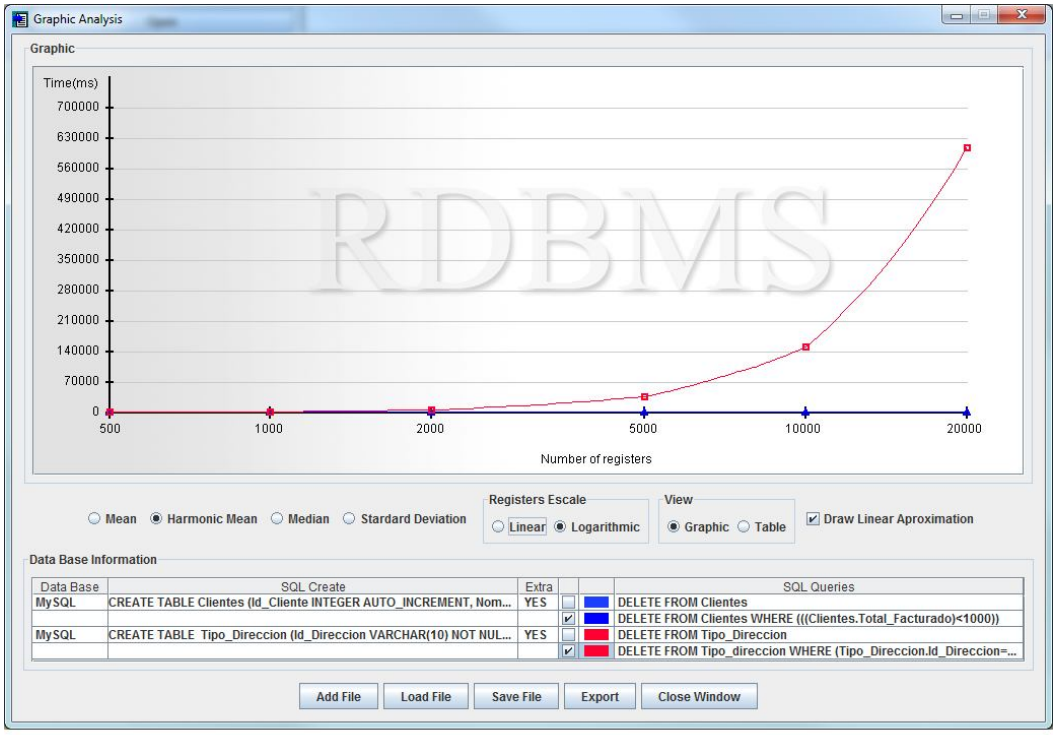
Para poder comprobar el rendimiento de los diferentes patrones de mapeo nos vamos a centrar en un SGDBR, se ha seleccionado MySQL dado lo equilibrado de su comportamiento.

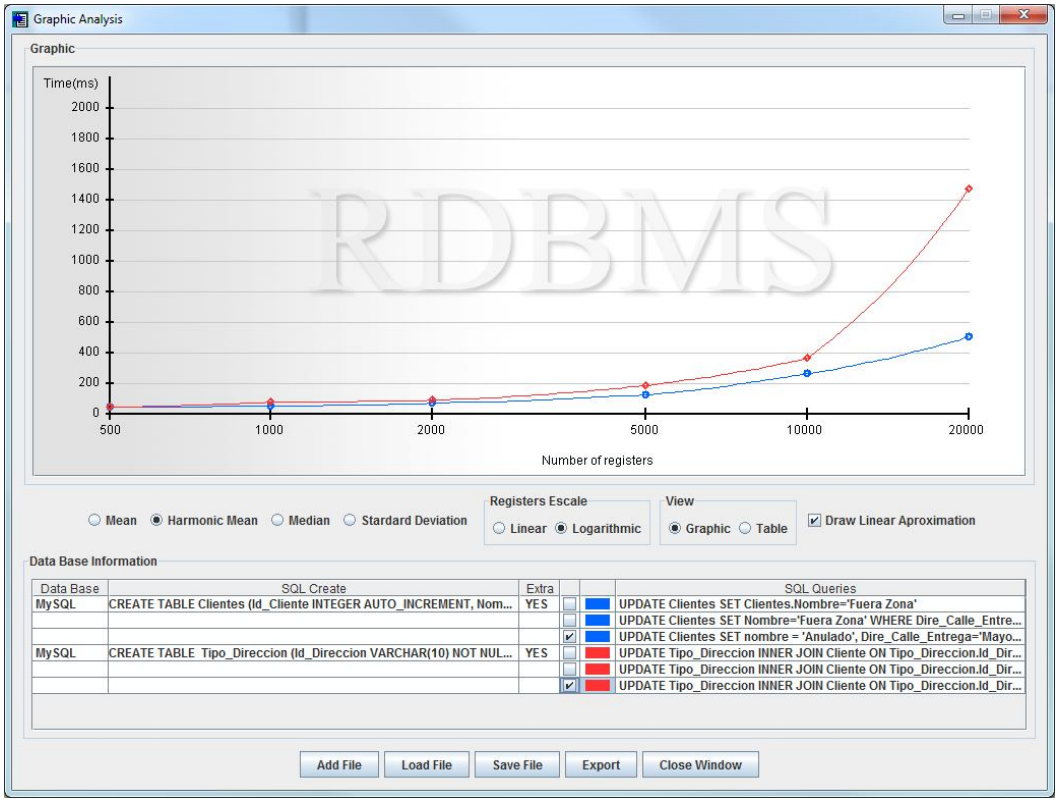
7.2.1 Agregación

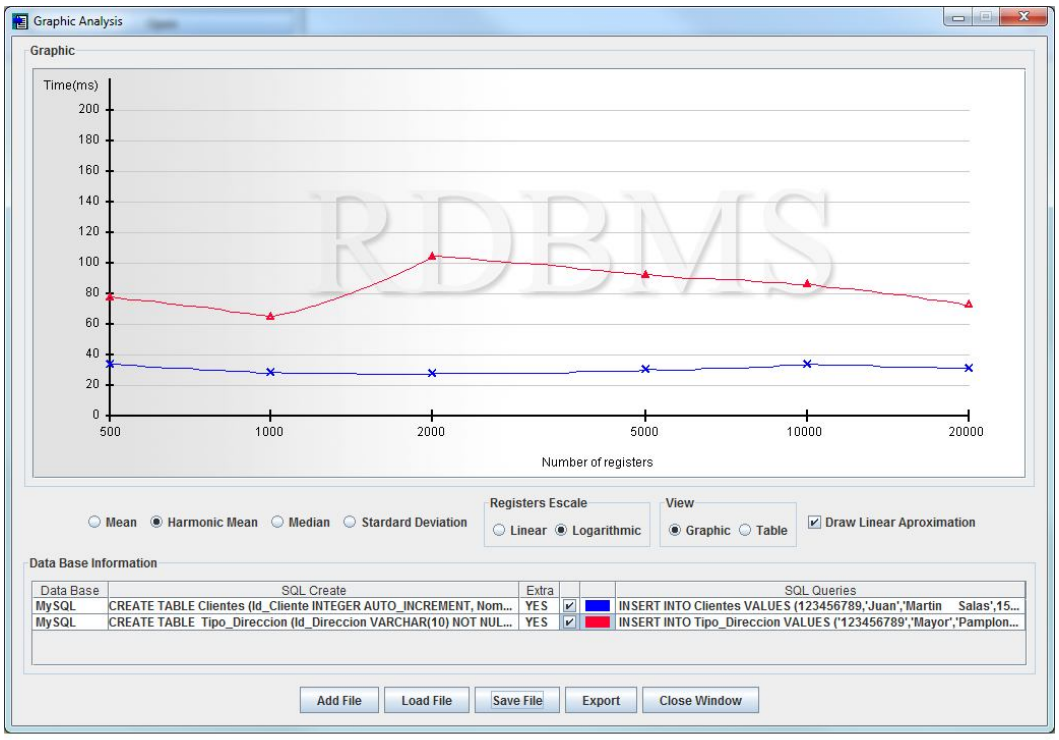
En las siguientes gráficas podemos visualizar los resultados obtenidos al aplicar los patrones de mapeo para la agregación, correspondiendo la línea azul al patrón de mapeo *Tabla Sencilla* y la línea roja la correspondiente al patrón de mapeo *Clave extranjera*.







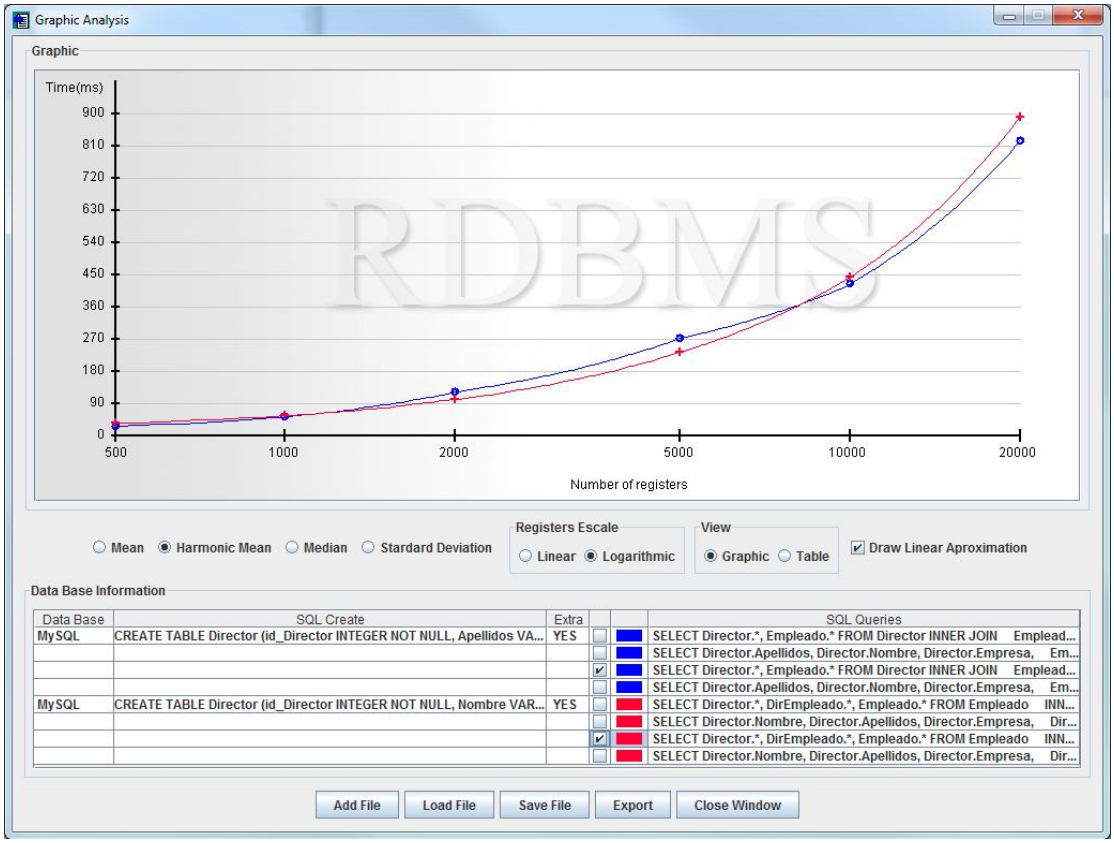


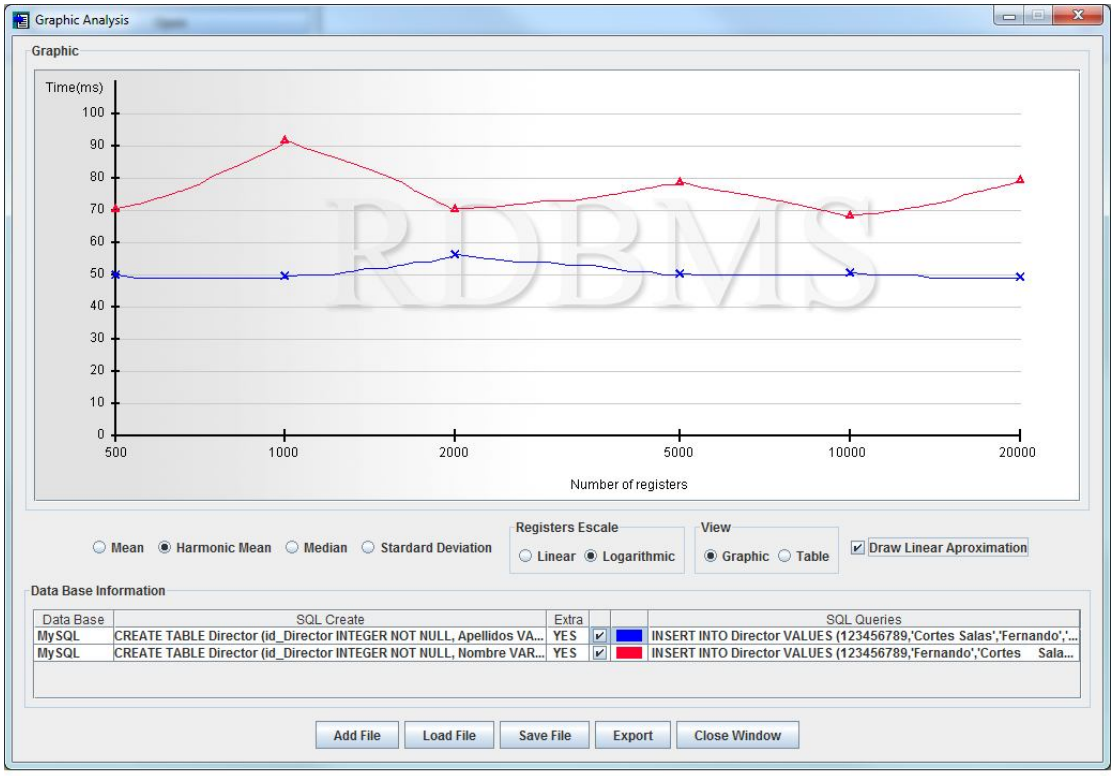


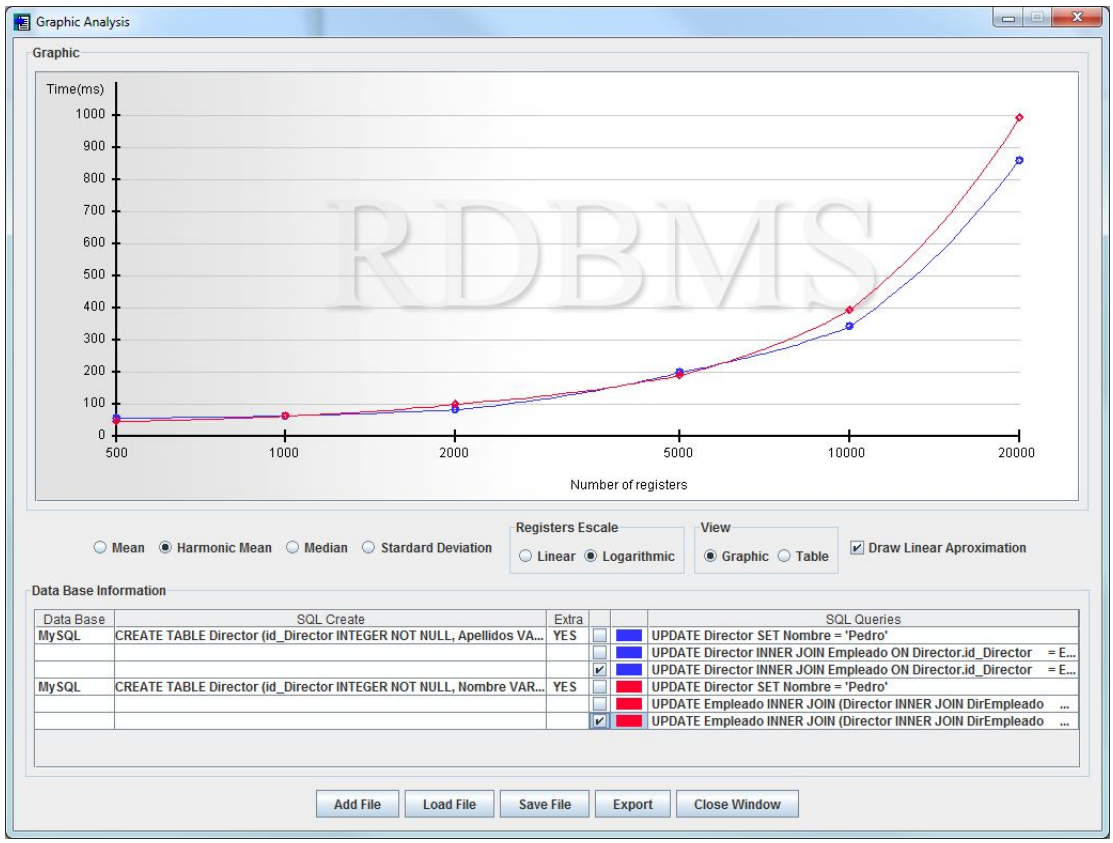
Observando las gráficas podemos comprobar cómo en términos de rendimiento el mapeo por *Tabla sencilla* es mayor, acentuándose las diferencias con el aumento de los registros sobre los que se trabaja. El tener que trabajar sobre varias tablas relacionadas aun cuando el tamaño de las mismas es reducido se cobra un fuerte impacto en el rendimiento. Como es de esperar la inserción de un nuevo objeto es mucho más lenta en el caso de mapeo por *Clave extranjera*, una vez más debido a tener que operar sobre varias tablas. Destaca como al no acceder al objeto agregado el rendimiento de ambos mapeos es el mismo. Podemos concluir que el comportamiento real del patrón de mapeo es igual al comportamiento teórico que se esperaba de él.

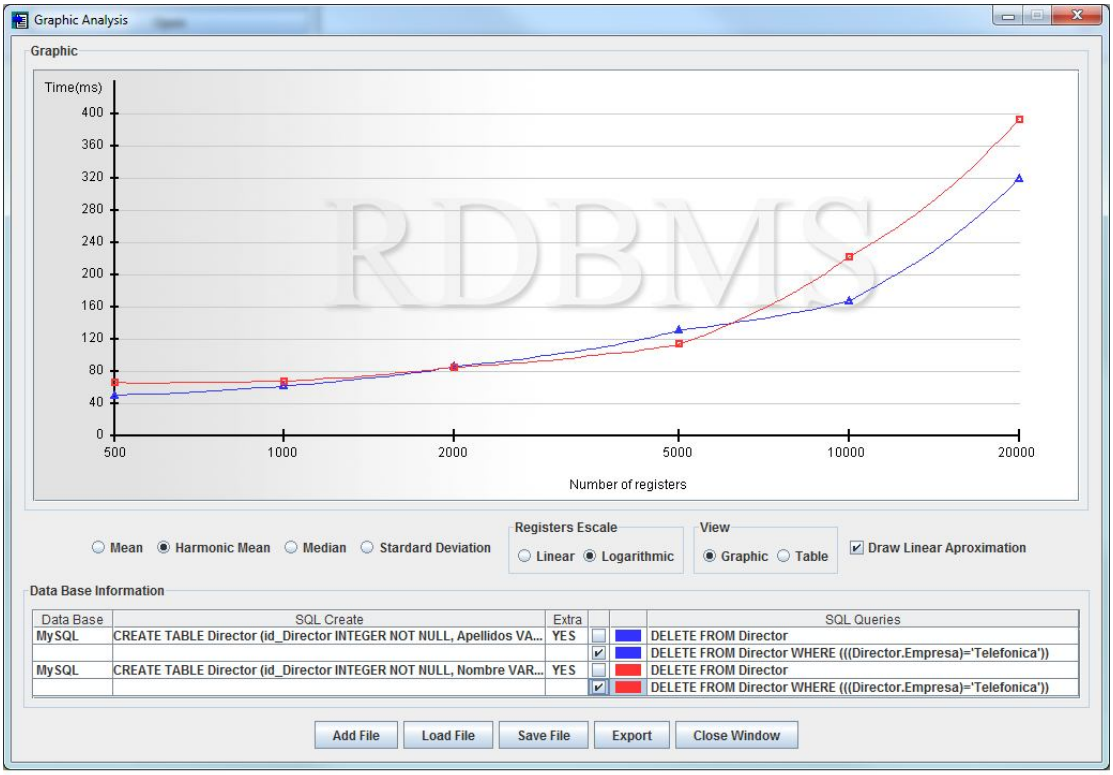
7.2.2 Asociación

En las siguientes gráficas podemos visualizar los resultados obtenidos al aplicar los patrones de mapeo de asociación. La línea azul la correspondiente al patrón de mapeo *Clave extranjera* y la línea roja la correspondiente al patrón de mapeo *Tabla de asociación*.





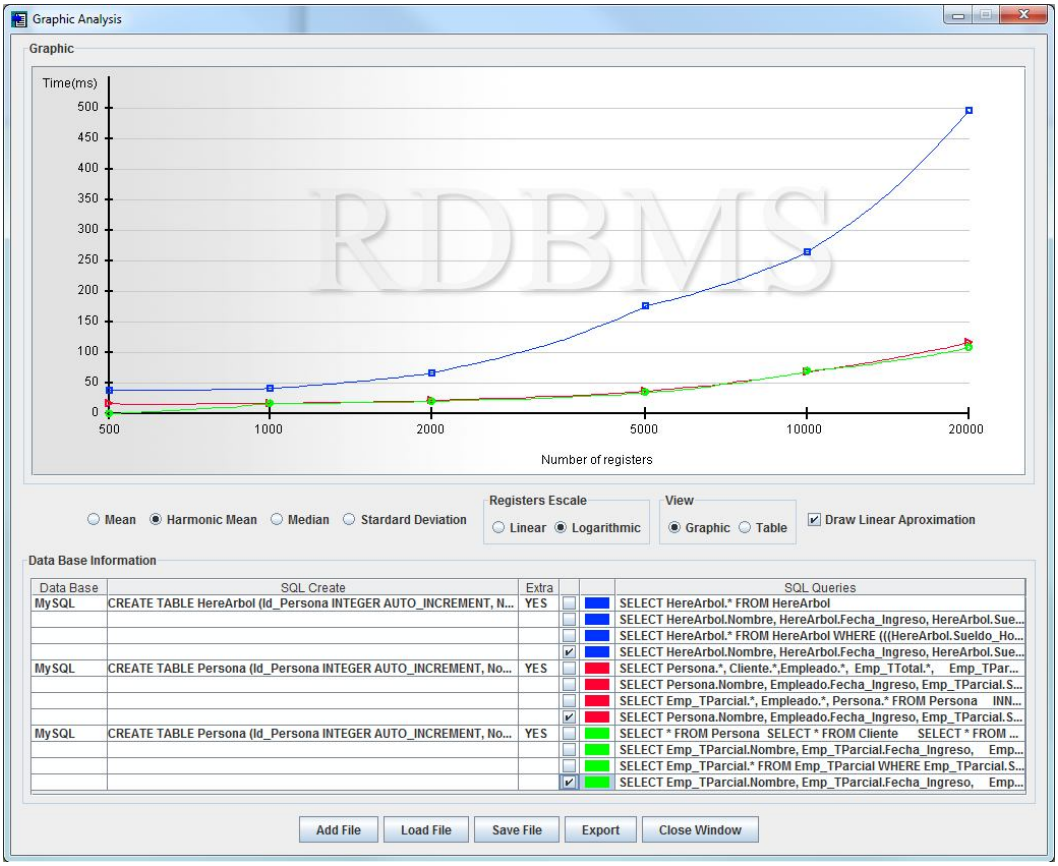


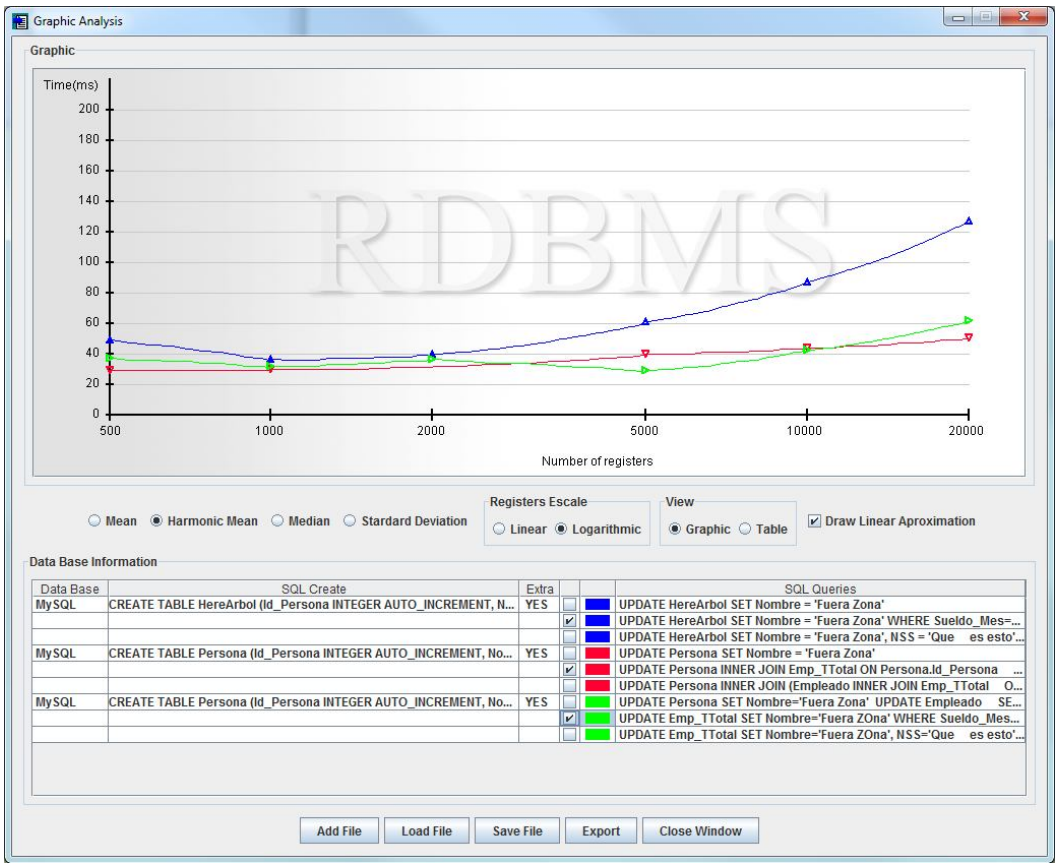


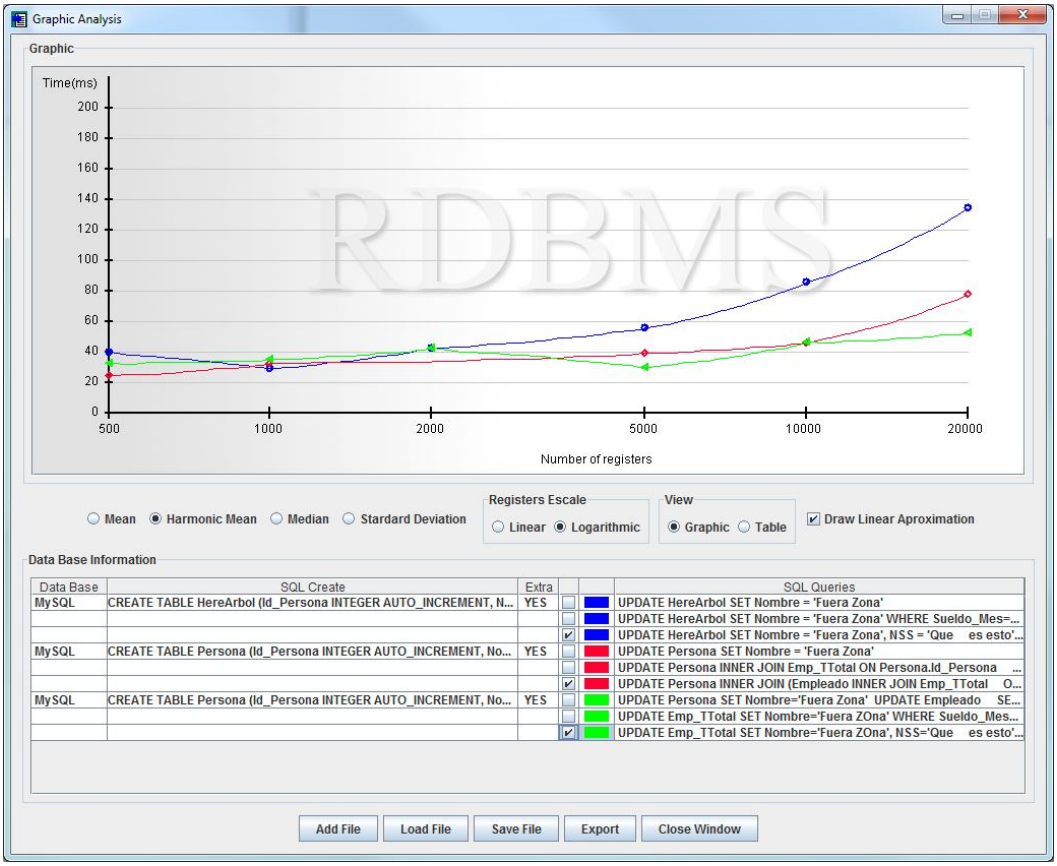
Se puede apreciar como los resultados obtenidos por ambos patrones en todos los tipos de petición son similares aunque algo mejores los obtenidos por el patrón *Clave extranjera*, una vez más los resultados reales son los esperados teóricamente.

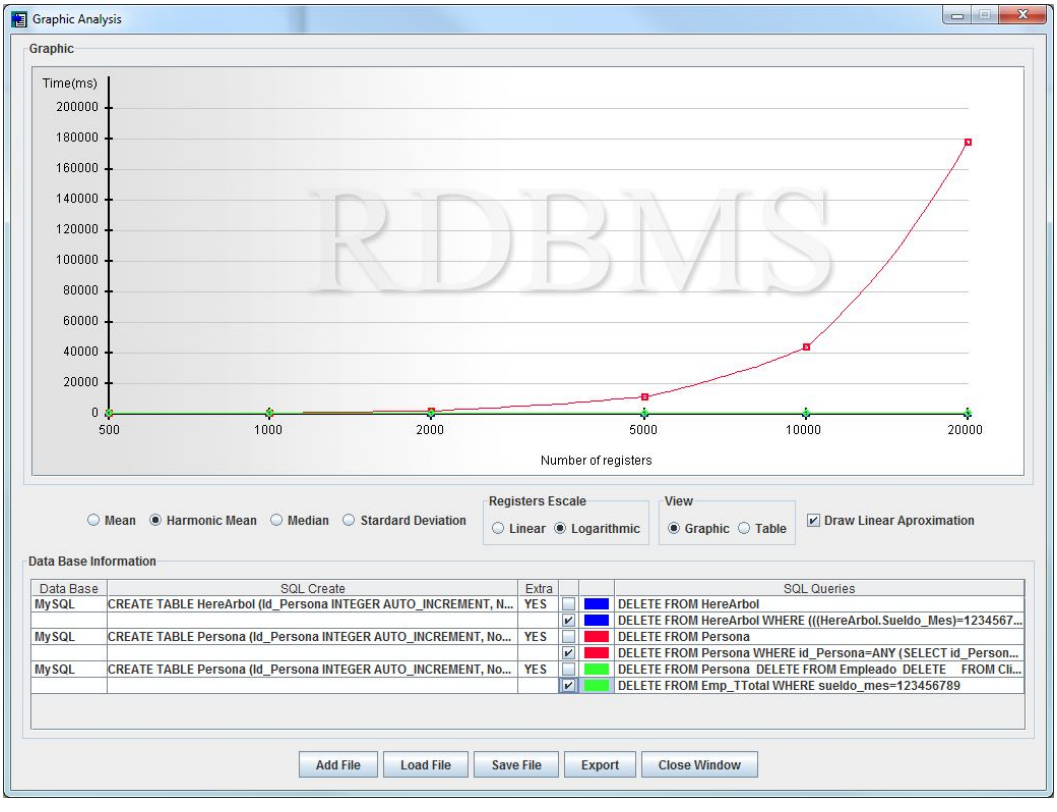
7.2.3 Herencia

En las siguientes gráficas podemos visualizar los resultados obtenidos, siendo la línea azul la correspondiente al patrón de mapeo *Herencia de árbol*, la línea roja la correspondiente al patrón de mapeo *Una clase una tabla* y la línea verde la correspondiente a *Una ruta de herencia una tabla*.









Podemos apreciar como en líneas generales el mapeo *Herencia de árbol* presenta el peor rendimiento en prácticamente todas las pruebas realizadas, seguramente debido a que a pesar de trabajar en únicamente una tabla esta contiene muchos registros y de gran tamaño. El mapeo *Una clase, una tabla* presenta un rendimiento irregular, se mantiene cercano al rendimiento ofrecido por *Una ruta de herencia, una tabla* salvo en determinadas condiciones donde su rendimiento se desploma al tener que ir asociando las diferentes tablas para poder obtener los objetos, algo que se agudiza conforme se va profundizando en la herencia. Podemos ver como se cumplen en cierto modo los resultados teóricos esperados, aunque con ciertos matices respecto al rendimiento de *Herencia de árbol*.

8. Conclusiones

Dada la naturaleza del proyecto de creación de una herramienta para la medición de rendimiento sobre diferentes SGBDR empleando diferentes mapeos de objetos vamos a dividir el apartado de conclusiones en tres áreas:

- Rendimientos de los diferentes SGBDR.
- Rendimiento de los diferentes mapeos de objetos.
- Herramienta.

8.1 Rendimiento de los diferentes SGBDR.

Realizando un análisis del conjunto de los resultados podemos observar que Access obtiene los peores resultados en todas las pruebas realizadas y con grandes diferencias respecto al resto de SGDBR, algo que en principio era de esperar y queda patente de este modo la naturaleza y el sector a que está orientado este sistema de gestión. Se puede apreciar como al trabajar sobre bases de datos de tamaño reducido, pese a que las diferencias con el resto de SGDBRs son relativamente grandes, los tiempos obtenidos son bajos, de modo que se podría utilizar en entornos donde el número de registros de la base de datos es reducido. También se observa como cuando las tablas de la base de datos tienen relaciones entre ellas los tiempos aumentan sustancialmente llegando a extremos de ser totalmente inoperativa, lo cual nos indica que su funcionamiento se limita a esquemas de tablas que no sean complejos. Destacar también que durante alguna de las pruebas se ha llegado a alcanzar el límite de 2GB de tamaño del fichero provocando el consiguiente error, lo cual nos vuelve a indicar que MS Access no puede ir más allá del uso en entornos personales o entornos profesionales sin grandes pretensiones, tanto de tamaño como de complejidad.

Respecto al resto de SGDBRs se ha de tener en cuenta que no se ha realizado ningún tipo de optimización ni ajustes del SGDBR para los diferentes esquemas. De modo que los rendimientos obtenidos se encuentran por debajo de los que se podrían llegar a obtener una vez configurados detalladamente los SGDBRs y si se hiciera un uso adecuado de índices para la aceleración sobre esquemas de múltiples tablas con diferentes relaciones. En cualquier caso se puede decir que, sin llegar a los límites de rendimiento, si que podemos realizar una comparación básica que puede orientar sobre el comportamiento de los SGDBRs.

En líneas generales, el comportamiento de los SGDBRs es similar entre unos y otros y como es de esperar los tiempos de respuesta aumentan junto con el número de registros. En los patrones de mapeo que hacen uso de una única tabla se aprecia como el comportamiento es lineal y con pequeñas variaciones en todos los SGDBRs, aunque destaca un comportamiento que presenta únicamente Oracle. Sin poder llegar a determinar el detonante de este comportamiento se aprecia un brusco aumento de los tiempos hasta alcanzar cierto número de registros y pasada la barrera se produce un considerable descenso de los tiempos. Podemos visualizar ese comportamiento en la figura 8.1.1. Al producirse únicamente con Oracle podemos descartar que tenga algo que ver con latencias de la red de comunicaciones. Tampoco podemos concretar si afecta a un tipo de consulta ya que se produce en una gran variedad de ellas, de modo que se puede deducir que se

debe al comportamiento interno que tiene Oracle que parece modificar algunos parámetros de funcionamiento en relación a alguna necesidad o tamaño de memoria. El valorar este tipo de comportamiento de un SGBDR tan complejo se encuentra fuera del alcance de este proyecto, sin embargo resaltar que la herramienta ha sido capaz de capturarlo.

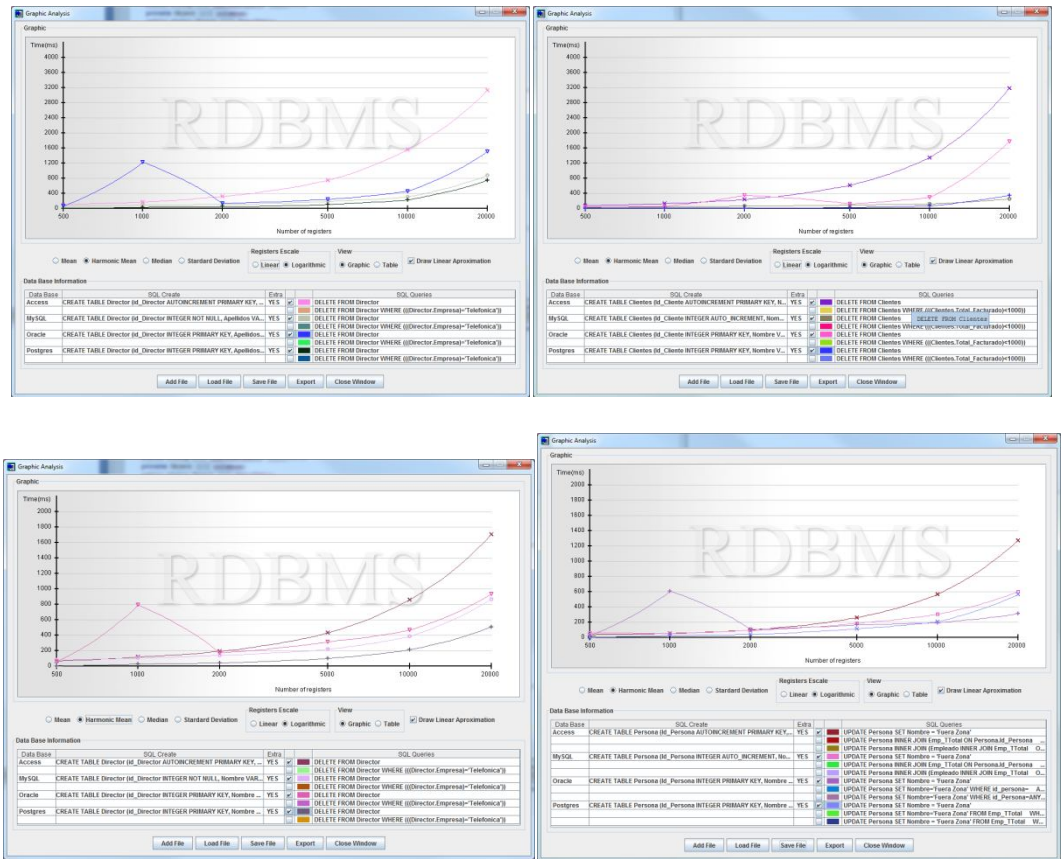


Figura 8.1.1: Irregularidades en Oracle.

Pese a la similitud del comportamiento de todos los SGDBRs se puede apreciar un patrón que se repite en la mayor parte de las pruebas realizadas. MySQL en general obtiene los mejores resultados en peticiones de consulta, pero se puede apreciar como decrece su ventaja al aumentar el tamaño de los registros incluso llegando a ser superado por PostgreSQL, aunque por muy poco. En este tipo de consultas Oracle sufre una importante penalización en rendimiento cuando el número de registros va aumentando en esquemas donde existen múltiples tablas relacionadas. PostgreSQL obtiene los mejores resultados en consultas de borrado independientemente del tamaño, numero de tablas y relaciones. Oracle obtiene ventajas sobre el resto en peticiones de actualización.

En función del tipo de mapeo no podemos concluir que un SGBDR destaque sobre el resto: unos se comportan mejor ante un tipo de consultas y otros ante otro tipo de consultas. Se podría determinar el uso de uno u otro SGBDR en función de los tipos de consulta predominantes, aunque debido a que las diferencias absolutas son muy pequeñas, quizás para decidirse por uno u otro SGBDR se debería valorar otros aspectos como pueden ser costes de implantación, mantenimiento, servicios, requerimientos o como se comportan ante situaciones de estrés con múltiples clientes accediendo simultáneamente al sistema.

8.2 Rendimiento de los diferentes patrones de mapeo.

En general se cumplen las estimaciones teóricas acerca del rendimiento sobre los diferentes patrones de mapeo, sin embargo vamos a detallar las conclusiones de cada patrón de mapeo.

Agregación:

> Observando las gráficas de la sección 6.2.1 podemos comprobar cómo, en términos de rendimiento, el mapeo por *Tabla sencilla* es mayor, acentuándose las diferencias con el aumento de los registros sobre los que se trabaja. El tener que trabajar sobre varias tablas relacionadas, aun cuando el tamaño de las mismas es reducido, se cobra un fuerte impacto en el rendimiento. Como es de esperar, la inserción de un nuevo objeto es mucho más lenta en el caso de mapeo *por Clave extranjera*, una vez más debido al hecho de tener que operar sobre varias tablas. Destaca como al no acceder al objeto agregado el rendimiento de ambos mapeos es el mismo. Podemos concluir que el comportamiento real del patrón de mapeo es igual al comportamiento teórico que se esperaba de él.

Asociación:

> Observando las gráficas de la sección 6.2.2 comprobamos como el rendimiento de ambos patrones es muy similar, aunque se puede apreciar como con el aumento del número de registros el rendimiento del mapeo de *Clave extranjera* es ligeramente superior debido a tener que acceder a menos tablas. El resultado experimental vuelve a coincidir con el rendimiento teórico.

Herencia:

> Observando las gráficas de la sección 6.2.2 comprobamos como en líneas generales el mapeo *Herencia de árbol* presenta el peor rendimiento en prácticamente todas las pruebas realizadas, seguramente debido a que a pesar de trabajar en únicamente una tabla esta contiene muchos registros y de gran tamaño. El mapeo *Una clase, una tabla* presenta un rendimiento irregular. Se mantiene cercano al rendimiento ofrecido por *Una ruta de herencia, una tabla* salvo en determinadas condiciones donde su rendimiento se desploma al tener que ir asociando las diferentes tablas para poder obtener los objetos, algo que se agudiza conforme se va profundizando en la herencia. Podemos ver como se cumplen en cierto modo los resultados teóricos esperados, aunque con ciertos matices respecto al rendimiento de *Herencia de árbol*.

8.3 La herramienta

La herramienta creada ha conseguido llegar más allá de todos los objetivos que tenía asignados. Nos ha permitido poder visualizar de una manera gráfica los patrones de mapeo de agregación, asociación y herencia, se ha podido realizar pruebas de rendimiento automatizadas sobre los diferentes patrones y sobre diferentes SGDBR. Con la misma herramienta se han podido visualizar y analizar todos los resultados obtenidos. Incluso se ha conseguido que nos permita modificar de una manera sencilla las pruebas automatizadas y además realizar pruebas definidas por el usuario en tiempo real.

8.4 Líneas futuras

La herramienta cuenta con una sólida base que permite realizar cualquier tipo de prueba de rendimiento sobre cualquier SGBDR, sin embargo sería interesante realizar pruebas sobre una base de datos orientada a objetos o aprovechar las mejoras relativas a objetos que se están introduciendo en las SGBDR . También se la podría dotar de la posibilidad del uso de herramientas de mapeo objeto-relacional (Hibernate, QuickDB...), extender el número de consultas para realizar un análisis más detallado del rendimiento de los diferentes patrones, variar el tamaño de los objetos a mapear para comprobar el impacto que esto tiene sobre el rendimiento de los diferentes patrones de mapeo y sobre los diferentes SGBDR. Sería muy interesante comprobar el comportamiento de los SGBDR cuando son atacados simultáneamente por varias herramientas y poder comprobar cómo afecta a los diferentes patrones de mapeo.

BIBLIOGRAFÍA

[1] http://www.alegsa.com.ar/Dic/paradigma.php
[2] http://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n
[3] http://en.wikipedia.org/wiki/Programming_paradigm
[4] http://www.monografias.com/trabajos16/paradigma/paradigma.shtml
[5] http://www.alegsaonline.com/art/13.php
[6] http://www.ulfix.net/programacion/conceptos-mainmenu-137/37-programacirientada-a-objetos
[7] http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos
[8] http://www.itq.edu.mx/vidatec/maestros/sis/mlopez/Tutorial/poo2.htm
[9] http://www.objectarchitects.de/ObjectArchitects/papers/Published/ZippedPapers/or06_proceedings.pdf
[10] http://www.objectarchitects.de/ObjectArchitects/papers/Published/ZippedPapers/mappings04.pdf
[11] http://office.microsoft.com/es-es/access-help/comparacion-entre-microsoft-jet-sql-y-ansi-sql-HP001032250.aspx
[12] http://www.mysql.com/
[13] http://www.postgresql.org/
[14] http://www.oracle.com/technetwork/java/overview-141217.html
[15] http://msdn.microsoft.com/en-us/library/ms710252%28VS.85%29.aspx
[16] http://aspen.ucs.indiana.edu/webtech/jdbc/overviewpaper/JDBCconn.html
[17] http://www.oracle.com/us/technologies/java/index.html
[18] http://msdn.microsoft.com/library/90h82b3x.aspx
[19] http://msdn.microsoft.com/es-es/beginner/cc305129%28en-us%29.aspx

Anexo 1. Pruebas realizadas.

A continuación se muestran los diferentes diagramas de clases con sus distintas relaciones y sus distintos mapeos a tablas relacionales.
Para la realización de las consultas sobre los mapeos relacionales se han seguido los siguientes patrones:

- Equivalencia de las consultas en todos los sistemas gestores. En caso de no poder realizar la misma consulta, se han utilizado consultas semejantes para conseguir el mismo objetivo.
- Equivalencia de objetivos en distintos mapeos, de tal manera que aunque el mapeo se implemente en una o en varias tablas relacionales el comportamiento de la consulta sea el mismo. Por ejemplo, realización de una consulta Select en un mapeo de herencia que se implementa en una única tabla y en varias tablas relacionadas.

1.1 Agregación

La figura 1.1.1 muestra el diagrama de clases de la agregación

Figura 1.1.1 Diagrama de clases agregación

La agregación tiene dos formas de mapeo, mapeo en tabla sencilla o única tabla y mapeo por clave extranjera. Se muestran dichos mapeos y las consultas que se han realizado sobre ellos.

- *Mapeo en tabla sencilla*: la figura 1.1.2 muestra la tabla relacional consecuencia del mapeo en tabla única de la agregación.

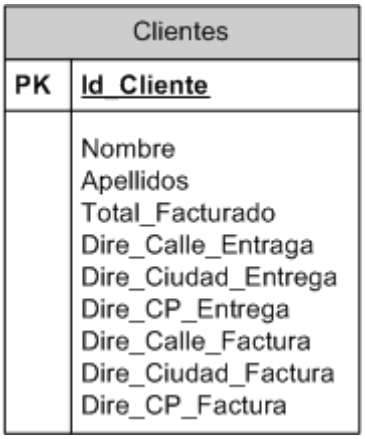


Figura 1.1.2 Mapeo en tabla sencilla

A continuación se muestran las sentencias SQL que se han realizado sobre este tipo de mapeo en los distintos sistemas gestores de bases de datos.

SELECT

- Select * from <tabla>
  - Se desea mostrar el valor de todos los campos de todos los registros y dado que se trabaja sobre una sola tabla la petición resulta sencilla y en todos los sistemas gestores de bases de datos se ha podido implementar la misma con el siguiente formato:
    - SELECT * FROM <*tabla*>
- Select <campos> from <tabla>
  - En este caso se desea que únicamente se muestren determinados campos de todos los registros y al igual que en el caso anterior, todos los sistemas gestores de bases de datos han podido realizar la misma consulta. Postgres no soporta nombrar un campo de la forma <tabla>.<campo>, pero dado que el resto de gestores soportan nombrar el campo sin describir la tabla se ha optado por el siguiente formato.
    - SELECT <$campo_1$>,<$campo_2$>,…,<$campo_n$> FROM <*tabla*>
- Select * from <tabla> where <condición>
  - Se muestran todos los campos de los registros que cumplen un cierto criterio. Dada la sencillez de la consulta todos los sistemas gestores de bases de datos han realizado la misma.
    - SELECT * FROM <*tabla*> WHERE <*campo*> < <*valor*>

- Select <campos> from <tabla> where <condición>
  - Se vuelve a mostrar tan sólo un número determinado de campos, pero añadiendo una condición. La misma consulta se realiza igual en todos los sistemas gestores de bases de datos.

    - SELECT <*campo*$_1$>,<*campo*$_2$>,…<*campo*$_n$> FROM <tabla> WHERE <*campo*> < <*valor*>

La tabla 1.1.1 muestra las consultas Select realizadas sobre los distintos sistemas gestores de bases de datos.

MS Access	Oracle	MySQL	Postgres
Select * from Clientes	Select * from Clientes	Select * from Clientes	Select * from Clientes
SELECT Id_Cliente, Nombre, Apellidos, Total_facturado, Dire_Calle_Entrega, Dire_CP_Entrega, Dire_Ciudad_Entrega from clientes	SELECT Id_Cliente, Nombre, Apellidos, Total_facturado, Dire_Calle_Entrega, Dire_CP_Entrega, Dire_Ciudad_Entrega from clientes	SELECT Id_Cliente, Nombre, Apellidos, Total_facturado, Dire_Calle_Entrega, Dire_CP_Entrega, Dire_Ciudad_Entrega from clientes	SELECT Id_Cliente, Nombre, Apellidos, Total_facturado, Dire_Calle_Entrega, Dire_CP_Entrega, Dire_Ciudad_Entrega from clientes
SELECT * FROM Clientes WHERE (total_facturado<1000000)	SELECT * FROM Clientes WHERE (total_facturado<1000000)	SELECT * FROM Clientes WHERE (total_facturado<1000000)	SELECT * FROM Clientes WHERE (total_facturado<1000000)
SELECT Id_Cliente, Nombre, Apellidos, Total_facturado, Dire_Calle_Entrega, Dire_CP_Entrega, Dire_Ciudad_Entrega FROM Clientes WHERE total_facturado <1000000	SELECT Id_Cliente, Nombre, Apellidos, Total_facturado, Dire_Calle_Entrega, Dire_CP_Entrega, Dire_Ciudad_Entrega FROM Clientes WHERE total_facturado <1000000	SELECT Id_Cliente, Nombre, Apellidos, Total_facturado, Dire_Calle_Entrega, Dire_CP_Entrega, Dire_Ciudad_Entrega FROM Clientes WHERE total_facturado <1000000	SELECT Id_Cliente, Nombre, Apellidos, Total_facturado, Dire_Calle_Entrega, Dire_CP_Entrega, Dire_Ciudad_Entrega FROM Clientes WHERE total_facturado <1000000

Tabla 1.1.1 Consultas Select tabla única

INSERT

- Insert into <tabla> values (<valor1>, <valor2>, … , <valorn>)

  - Se procede a la inserción de un registro en una tabla. En todos los sistemas gestores de bases de datos se ha utilizado la misma consulta SQL. Al tratarse de una tabla única se ha podido insertar todos los valores de los campos en una sóla consulta con el siguiente formato.

    - INSERT INTO <*tabla*> VALUES (<*valor*$_1$>, <*valor*$_2$>, … <*valor*$_n$>)

La tabla 1.1.2 muestra las consultas Insert realizadas sobre los distintos sistemas gestores de bases de datos.

MS Access	Oracle	MySQL	Postgres
INSERT INTO Clientes VALUES (123456789,'Juan','Martin Salas',1500,'Mayor','Pamplona','31115','Menor','Tudela','31000')	INSERT INTO Clientes VALUES (123456789,'Juan','Martin Salas',1500,'Mayor','Pamplona','31115','Menor','Tudela','31000')	INSERT INTO Clientes VALUES (123456789,'Juan','Martin Salas',1500,'Mayor','Pamplona','31115','Menor','Tudela','31000')	INSERT INTO Clientes VALUES (123456789,'Juan','Martin Salas',1500,'Mayor','Pamplona','31115','Menor','Tudela','31000')

Tabla 1.1.2 Consultas Insert tabla única

UPDATE

- Update <tabla> set <campo> = <valor>
  - Se trata de una consulta SQL que se limita a actualizar el valor de un campo sobre todos los registros de la tabla.
    - UPDATE <*tabla*> SET <*campo*> = <*valor*>
- Update <tabla> set <campo> = <valor> where <condición>
  - En esta ocasión se añade un criterio de actualización. Al trabajar sobre una sola tabla el formato de la petición es muy sencillo.
    - UPDATE <*tabla*> SET <$campo_1$> = <$valor_1$> WHERE <$campo_2$> = <$valor_2$>
- Update <tabla> set <campos> = <valores> where <condición>
  - Al igual que en los casos anteriores la consulta es muy sencilla y se limita a actualizar dos campos de un registro cuando un campo de éste cumple una condición.
    - UPDATE <*tabla*> SET <$campo_m$>= <$valor_m$> , <$campo_n$> = <$valor_n$> WHERE <$campo_x$> < <$valor_x$>

La tabla 1.1.3 muestra las consultas Update realizadas sobre los distintos sistemas gestores de bases de datos.

MS Access	Oracle	MySQL	Postgres
UPDATE Clientes SET Clientes.Nombre='Fuera Zona'	UPDATE Clientes SET Clientes.Nombre='Fuera Zona'	UPDATE Clientes SET Clientes.Nombre='Fuera Zona'	UPDATE Clientes SET Nombre='Fuera Zona'
UPDATE Clientes SET Nombre='Fuera Zona' WHERE Dire_Calle_Entrega='Mayor'	UPDATE Clientes SET Nombre='Fuera Zona' WHERE Dire_Calle_Entrega='Mayor'	UPDATE Clientes SET Nombre='Fuera Zona' WHERE Dire_Calle_Entrega='Mayor'	UPDATE Clientes SET Nombre='Fuera Zona' WHERE Dire_Calle_Entrega='Mayor'
UPDATE Clientes SET	UPDATE Clientes SET	UPDATE Clientes SET	UPDATE Clientes SET

nombre = 'Anulado', Dire_Calle_Entrega='Mayor' WHERE total_facturado<200	nombre = 'Anulado', Dire_Calle_Entrega='Mayor' WHERE total_facturado<200	nombre = 'Anulado', Dire_Calle_Entrega='Mayor' WHERE total_facturado<200	nombre = 'Anulado', Dire_Calle_Entrega='Mayor' WHERE total_facturado<200

Tabla 1.1.3 Consultas Update tabla única

DELETE

- Delete * from <tabla>
  - Se procede al borrado de todos los registros. Dado que el '*' no es soportado por todos los gestores de bases de datos, se ha eliminado de la sentencia.
    - DELETE FROM <*tabla*>
- Delete * where <condicion>
  - En este caso se añade un criterio de eliminación y al igual que en el caso anterior también se ha conseguido una consulta compatible con todos los sistemas gestores eliminado el carácter '*'.
    - DELETE FROM <*tabla*> WHERE <*tabla*>.<$campo_x$> <*valor*>

La tabla 1.1.4 muestra las consultas Delete realizadas sobre los distintos sistemas gestores de bases de datos.

MS Access	Oracle	MySQL	Postgres
DELETE FROM Clientes	DELETE FROM Clientes	DELETE FROM Clientes	DELETE FROM Clientes
DELETE FROM Clientes WHERE (((Clientes.Total_Facturado)<1000))	DELETE FROM Clientes WHERE (((Clientes.Total_Facturado)<1000))	DELETE FROM Clientes WHERE (((Clientes.Total_Facturado)<1000))	DELETE FROM Clientes WHERE (((Clientes.Total_Facturado)<1000))

Tabla 1.1.4 Consultas Delete tabla única

- *Mapeo por clave extranjera*: la figura 1.1.3 muestra las tablas relacionales consecuencia del mapeo por clave extranjera de la agregación.

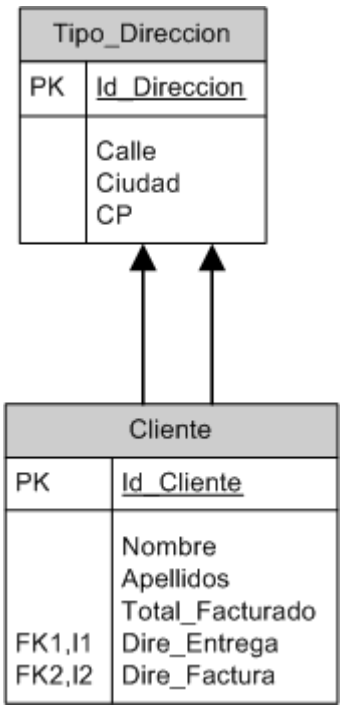


Figura 1.1.3 Mapeo por clave extranjera

A continuación se muestran las sentencias SQL que se han realizado sobre este tipo de mapeo en los distintos sistemas gestores de bases de datos.

SELECT

- Select * from <tabla>
  - El proceso que se lleva a cabo para realizar esta consulta consiste en la creación de una tabla intermedia (consulta de creación de tabla) que contiene a dos tablas (tablaTipo_Direccion y tabla Cliente relacionadas por Dire_Entrega) y posteriormente se realiza una consulta sobre esta tabla intermedia y la tabla Cliente mediante el uso de INNER JOIN, relacionadas con Dire_Factura. Dado que esta consulta tiene un cierto grado de complejidad no se ha conseguido crear una consulta SQL que sea soportada por todos los sistemas gestoresa de bases de datos, debido en gran medida a la creación de la tabla intermedia ya que cada sistema gestor tiene su propia sintaxis.
    - MS Access:
      - SELECT <*tabla*$_1$>.* <*tabla*$_2$>.* INTO <*tabla*$_3$> FROM <*tabla*$_1$> INNER JOIN <*tabla*$_2$> ON <tabla$_1$>.<campo$_{1,1}$>=<tabla$_2$>.<*campo*$_{1,1}$>;
      - SELECT <*tabla*$_3$>.<*campo*$_{3,1}$>, <*tabla*$_3$>.<*campo*$_{3,2}$>, … ,<*tabla*$_3$>.<*campo*$_{3,n}$>, <*tabla*$_2$>.<*campo*$_{2,1}$>, <*tabla*$_2$>.<*campo*$_{2,2}$>, … ,<*tabla*$_2$>.<*campo*$_{2,n}$> FROM <*tabla*$_2$> INNER JOIN <*tabla*$_3$> ON <*tabla*$_2$>.<*campo*$_{2,1}$> = <*tabla*$_2$>.<*campo*$_{3,1}$>
    - Oracle:

- CREATE TABLE <*tabla*$_3$> AS SELECT <*tabla*$_2$>.* <*tabla*$_1$>.* FROM <*tabla*$_1$> INNER JOIN <*tabla*$_2$> ON <*tabla*$_2$>.<*campo*$_{2,1}$> = <*tabla*$_1$>.<*campo*$_{1,1}$>
- SELECT <*tabla*$_3$>.<*campo*$_{3,1}$> <*tabla*$_3$>.<*campo*$_{3,2}$>, … ,<*tabla*$_3$>.<*campo*$_{3,n}$>, <*tabla*$_2$>.<*campo*$_{2,1}$>, <*tabla*$_2$>.<*campo*$_{2,2}$>, …, <*tabla*$_2$>.<*campo*$_{2,m}$> FROM <*tabla*$_2$> INNER JOIN <*tabla*$_3$> ON <*tabla*$_2$>.<*campo*$_{2,1}$> = <*tabla*$_3$>.<*campo*$_{3,1}$>

- MySQL:
  - CREATE TABLE <tabla3> SELECT <*tabla2*>.* , <*tabla1*>.* FROM <*tabla*$_1$> INNER JOIN <*tabla*$_2$> ON <*tabla*$_2$>.<*campo*$_{2,1}$> = <*tabla*$_1$>.<*campo*$_{1,1}$>
  - SELECT <*tabla*$_3$>.<*campo*$_{3,1}$> <*tabla*$_3$>.<*campo*$_{3,2}$>, … ,<*tabla*$_3$>.<*campo*$_{3,n}$>, <*tabla*$_2$>.<*campo*$_{2,1}$>, <*tabla*$_2$>.<*campo*$_{2,2}$>, …, <*tabla*$_2$>.<*campo*$_{2,m}$> FROM <*tabla*$_2$> INNER JOIN <*tabla*$_3$> ON <*tabla*$_2$>.<*campo*$_{2,1}$> = <*tabla*$_3$>.<*campo*$_{3,1}$>
- Postgres:
  - CREATE TABLE <*tabla*$_3$> AS SELECT <*tabla*$_2$>.* <*tabla*$_1$>.* FROM <*tabla*$_1$> INNER JOIN <*tabla*$_2$> ON <*tabla*$_2$>.<*campo*$_{2,1}$> = <*tabla*$_1$>.<*campo*$_{1,1}$>
  - SELECT <*tabla*$_3$>.<*campo*$_{3,1}$> <*tabla*$_3$>.<*campo*$_{3,2}$>, … ,<*tabla*$_3$>.<*campo*$_{3,n}$>, <*tabla*$_2$>.<*campo*$_{2,1}$>, <*tabla*$_2$>.<*campo*$_{2,2}$>, …, <*tabla*$_2$>.<*campo*$_{2,m}$> FROM <*tabla*$_2$> INNER JOIN <*tabla*$_3$> ON <*tabla*$_2$>.<*campo*$_{2,1}$> = <*tabla*$_3$>.<*campo*$_{3,1}$>

- Select <campos> from <tabla>
  -
  - Se ha conseguido una consulta compatible con todos los sistemas gestores de bases de datos. El procedimiento ha consistido en realizar un SELECT que relaciona dos tablas utilizando INNER JOIN. El formato de la consulta resultante es:

    - SELECT <*tabla*$_1$>.<*campo*$_u$>, <*tabla*$_1$>.<*campo*$_v$>, …., <*tabla*$_1$>.<*campos*$_z$>, <*tabla*$_2$>.<*campo*$_i$>, <*tabla*$_2$>.<*campo*$_j$>, …, <*tabla*$_2$>.<*campo*$_h$> FROM <*tabla*$_1$> INNER JOIN <*tabla*$_2$> ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_2$>.<*campo*$_1$>

- Select * from <tabla> where <condición>
  - Para implementar esta consulta se han utilizado el mismo proceso que en el caso anterior (“*Select * from <tabla>*”)
    - MS Access:
      - SELECT <*tabla*$_1$>.* <*tabla*$_2$>.* INTO <*tabla*$_3$> FROM <*tabla*$_1$> INNER JOIN <*tabla*$_2$> ON <*tabla*$_1$>.<*campo*$_{1,1}$> = <*tabla*$_2$>.<*campo*$_{2,1}$>;

- SELECT <*tabla*$_3$>.<*campo*$_{3,1}$>, <*tabla*$_3$>.<*campo*$_{3,2}$>, … , <*tabla*$_3$>.<*campo*$_{3,n}$>, <*tabla*$_2$>.<*campo*$_{2,1}$>, <*tabla*$_2$>.<*campo*$_{2,2}$>, … , <*tabla*$_2$>.<*campo*$_{2,n}$> FROM <*tabla*$_2$> INNER JOIN <*tabla*$_3$> ON <*tabla*$_2$>.<*campo*$_{2,1}$> = <*tabla*$_3$>.<*campo*$_{3,1}$> WHERE <*tabla*$_3$>.<*campo*$_x$> < <*valor*>

- Oracle:
  - CREATE TABLE <*tabla*$_3$> AS SELECT <*tabla*$_1$>.* <*tabla*$_2$>.* FROM <*tabla*$_1$> INNER JOIN <*tabla*$_2$> ON <*tabla*$_1$>.<*campo*$_{1,1}$> = <*tabla*$_2$>.<*campo*$_{2,1}$>;
  - SELECT <*tabla*$_3$>.<*campo*$_{3,1}$>, <*tabla*$_3$>.<*campo*$_{3,2}$>, … , <*tabla*$_3$>.<*campo*$_{3,n}$>, <*tabla*$_2$>.<*campo*$_{2,1}$>, <*tabla*$_2$>.<*campo*$_{2,2}$>, … , <*tabla*$_2$>.<*campo*$_{2,n}$> FROM <*tabla*$_2$> INNER JOIN <*tabla*$_3$> ON <*tabla*$_2$>.<*campo*$_{2,1}$> = <*tabla*$_3$>.<*campo*$_{3,1}$> WHERE <*tabla*$_3$>.<*campo*$_x$> < <*valor*>
- MySQL:
  - CREATE TABLE <*tabla*$_3$> SELECT <*tabla*$_1$>.* <*tabla*$_2$>.* FROM <*tabla*$_1$> INNER JOIN <*tabla*$_2$> ON <*tabla*$_1$>.<*campo*$_{1,1}$> = <*tabla*$_2$>.<*campo*$_{2,1}$>;
  - SELECT <*tabla*$_3$>.<*campo*$_{3,1}$>, <*tabla*$_3$>.<*campo*$_{3,2}$>, … , <*tabla*$_3$>.<*campo*$_{3,n}$>, <*tabla*$_2$>.<*campo*$_{2,1}$>, <*tabla*$_2$>.<*campo*$_{2,2}$>, … , <*tabla*$_2$>.<*campo*$_{2,n}$> FROM <*tabla*$_2$> INNER JOIN <*tabla*$_3$> ON <*tabla*$_2$>.<*campo*$_{2,1}$> = <*tabla*$_3$>.<*campo*$_{3,1}$> WHERE <*tabla*$_3$>.<*campo*$_x$> < <*valor*>
- Postgres:
  - CREATE TABLE <*tabla*$_3$> AS SELECT <*tabla*$_1$>.* <*tabla*$_2$>.* FROM <*tabla*$_1$> INNER JOIN <*tabla*$_2$> ON <*tabla*$_1$>.<*campo*$_{1,1}$> = <*tabla*$_2$>.<*campo*$_{2,1}$>;
  - SELECT <*tabla*$_3$>.<*campo*$_{3,1}$>, <*tabla*$_3$>.<*campo*$_{3,2}$>, … , <*tabla*$_3$>.<*campo*$_{3,n}$>, <*tabla*$_2$>.<*campo*$_{2,1}$>, <*tabla*$_2$>.<*campo*$_{2,2}$>, … , <*tabla*$_2$>.<*campo*$_{2,n}$> FROM <*tabla*$_2$> INNER JOIN <*tabla*$_3$> ON <*tabla*$_2$>.<*campo*$_{2,1}$> = <*tabla*$_3$>.<*campo*$_{3,1}$> WHERE <*tabla*$_3$>.<*campo*$_x$> < <*valor*>

- Select <campos> from <tabla> where <condición>
  - Esta consulta SQL se ha realizado de la misma manera en todos los sistemas gestores de bases de datos. Se trata de un SELECT sobre dos tablas utilizando INNER JOIN. El formato de la consulta es el siguiente:

    - SELECT <*tabla*$_1$>.<*campo*$_n$>, <*tabla*$_1$>.<*campo*$_{1,v}$>, … , <*tabla1*>.<*campo*$_{1,z}$>, <*tabla2*>.<*campo*$_{2,i}$>, <*tabla*$_2$>.<*campo*$_{2,j}$>, … , <*tabla*$_2$>.<*campo*$_h$> FROM <*tabla*$_1$> INNER JOIN <*tabla*$_2$> WHERE <*tabla*$_1$>.<*campo*$_x$> < <*valor*>

La tabla 1.1.5 muestra las consultas Select realizadas sobre los distintos sistemas gestores de bases de datos.

MS Access	Oracle	MySQL	Postgres
SELECT Tipo_Direccion.*, Cliente.* INTO cliente1 FROM Tipo_Direccion INNER JOIN Cliente ON Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega; SELECT cliente1.Id_Cliente, cliente1.Nombre, cliente1.Apellidos, cliente1.Total_Facturado, Tipo_Direccion.Calle, Tipo_Direccion.Ciudad, Tipo_Direccion.CP, cliente1.Calle, cliente1.CP, cliente1.Ciudad FROM Tipo_Direccion INNER JOIN cliente1 ON Tipo_Direccion.Id_Direccion = cliente1.Dire_Factura	CREATE TABLE cliente1 AS SELECT Tipo_Direccion.*, Cliente.* FROM Tipo_Direccion INNER JOIN Cliente ON Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega SELECT cliente1.Id_Cliente, cliente1.Nombre, cliente1.Apellidos, cliente1.Total_Facturado, Tipo_Direccion.Calle, Tipo_Direccion.Ciudad, Tipo_Direccion.CP, cliente1.Calle, cliente1.CP, cliente1.Ciudad FROM Tipo_Direccion INNER JOIN cliente1 ON Tipo_Direccion.Id_Direccion = cliente1.Dire_Factura	CREATE TABLE cliente1 SELECT Tipo_Direccion.*, Cliente.* FROM Tipo_Direccion INNER JOIN Cliente ON Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega; SELECT cliente1.Id_Cliente, cliente1.Nombre, cliente1.Apellidos, cliente1.Total_Facturado, Tipo_Direccion.Calle, Tipo_Direccion.Ciudad, Tipo_Direccion.CP, cliente1.Calle, cliente1.CP, cliente1.Ciudad FROM Tipo_Direccion INNER JOIN cliente1 ON Tipo_Direccion.Id_Direccion = cliente1.Dire_Factura	CREATE TABLE cliente1 AS SELECT Tipo_Direccion.*, Cliente.* FROM Tipo_Direccion INNER JOIN Cliente ON Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega SELECT cliente1.Id_Cliente, cliente1.Nombre, cliente1.Apellidos, cliente1.Total_Facturado, Tipo_Direccion.Calle, Tipo_Direccion.Ciudad, Tipo_Direccion.CP, cliente1.Calle, cliente1.CP, cliente1.Ciudad FROM Tipo_Direccion INNER JOIN cliente1 ON Tipo_Direccion.Id_Direccion = cliente1.Dire_Factura
SELECT Cliente.Id_Cliente, Cliente.Nombre, Cliente.Apellidos, Cliente.Total_Facturado, Tipo_Direccion.Calle, Tipo_Direccion.CP, Tipo_Direccion.Ciudad FROM Tipo_Direccion INNER JOIN Cliente ON Tipo_Direccion.Id_Direccion = Cliente.Dire_Factura	SELECT Cliente.Id_Cliente, Cliente.Nombre, Cliente.Apellidos, Cliente.Total_Facturado, Tipo_Direccion.Calle, Tipo_Direccion.CP, Tipo_Direccion.Ciudad FROM Tipo_Direccion INNER JOIN Cliente ON Tipo_Direccion.Id_Direccion = Cliente.Dire_Factura	SELECT Cliente.Id_Cliente, Cliente.Nombre, Cliente.Apellidos, Cliente.Total_Facturado, Tipo_Direccion.Calle, Tipo_Direccion.CP, Tipo_Direccion.Ciudad FROM Tipo_Direccion INNER JOIN Cliente ON Tipo_Direccion.Id_Direccion = Cliente.Dire_Factura	SELECT Cliente.Id_Cliente, Cliente.Nombre, Cliente.Apellidos, Cliente.Total_Facturado, Tipo_Direccion.Calle, Tipo_Direccion.CP, Tipo_Direccion.Ciudad FROM Tipo_Direccion INNER JOIN Cliente ON Tipo_Direccion.Id_Direccion = Cliente.Dire_Factura
SELECT Tipo_Direccion.*, Cliente.* INTO cliente1 FROM Tipo_Direccion INNER JOIN Cliente ON Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega;	CREATE TABLE cliente1 AS SELECT Tipo_Direccion.*, Cliente.* FROM Tipo_Direccion INNER JOIN Cliente ON Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega	CREATE TABLE cliente1 SELECT Tipo_Direccion.*, Cliente.* FROM Tipo_Direccion INNER JOIN Cliente ON Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega;	CREATE TABLE cliente1 AS SELECT Tipo_Direccion.*, Cliente.* FROM Tipo_Direccion INNER JOIN Cliente ON Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega

SELECT cliente1.Id_Cliente, cliente1.Nombre, cliente1.Apellidos, cliente1.Total_Facturado , Tipo_Direccion.Calle, Tipo_Direccion.Ciudad, Tipo_Direccion.CP, cliente1.Calle, cliente1.CP, cliente1.Ciudad FROM Tipo_Direccion INNER JOIN cliente1 ON Tipo_Direccion.Id_Direccion = cliente1.Dire_Factura WHERE Cliente1.Total_Facturado<1000000	SELECT cliente1.Id_Cliente, cliente1.Nombre, cliente1.Apellidos, cliente1.Total_Facturado , Tipo_Direccion.Calle, Tipo_Direccion.Ciudad, Tipo_Direccion.CP, cliente1.Calle, cliente1.CP, cliente1.Ciudad FROM Tipo_Direccion INNER JOIN cliente1 ON Tipo_Direccion.Id_Direccion = cliente1.Dire_Factura WHERE Cliente1.Total_Facturado<1000000	SELECT cliente1.Id_Cliente, cliente1.Nombre, cliente1.Apellidos, cliente1.Total_Facturado , Tipo_Direccion.Calle, Tipo_Direccion.Ciudad, Tipo_Direccion.CP, cliente1.Calle, cliente1.CP, cliente1.Ciudad FROM Tipo_Direccion INNER JOIN cliente1 ON Tipo_Direccion.Id_Direccion = cliente1.Dire_Factura WHERE Cliente1.Total_Facturado<1000000	SELECT cliente1.Id_Cliente, cliente1.Nombre, cliente1.Apellidos, cliente1.Total_Facturado , Tipo_Direccion.Calle, Tipo_Direccion.Ciudad, Tipo_Direccion.CP, cliente1.Calle, cliente1.CP, cliente1.Ciudad FROM Tipo_Direccion INNER JOIN cliente1 ON Tipo_Direccion.Id_Direccion = cliente1.Dire_Factura WHERE Cliente1.Total_Facturado<1000000
SELECT Cliente.Id_Cliente, Cliente.Nombre, Cliente.Apellidos, Cliente.Total_Facturado, Tipo_Direccion.Calle, Tipo_Direccion.CP, Tipo_Direccion.Ciudad FROM Tipo_Direccion INNER JOIN Cliente ON (Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega) WHERE ((Cliente.Total_Facturado)<1000000)	SELECT Cliente.Id_Cliente, Cliente.Nombre, Cliente.Apellidos, Cliente.Total_Facturado, Tipo_Direccion.Calle, Tipo_Direccion.CP, Tipo_Direccion.Ciudad FROM Tipo_Direccion INNER JOIN Cliente ON (Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega) WHERE ((Cliente.Total_Facturado)<1000000)	SELECT Cliente.Id_Cliente, Cliente.Nombre, Cliente.Apellidos, Cliente.Total_Facturado, Tipo_Direccion.Calle, Tipo_Direccion.CP, Tipo_Direccion.Ciudad FROM Tipo_Direccion INNER JOIN Cliente ON (Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega) WHERE ((Cliente.Total_Facturado)<1000000)	SELECT Cliente.Id_Cliente, Cliente.Nombre, Cliente.Apellidos, Cliente.Total_Facturado, Tipo_Direccion.Calle, Tipo_Direccion.CP, Tipo_Direccion.Ciudad FROM Tipo_Direccion INNER JOIN Cliente ON (Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega) WHERE ((Cliente.Total_Facturado)<1000000)

Tabla 1.1.5 Consultas Select por clave extranjera

INSERT:

- Insert into <tabla> values (<valor1>,<valor2>, …, <$valor_n$>)
  - Al tratarse de un mapeo de agregación por clave extranjera sobre dos tablas, se han tenido que encadenar tres consultas INSERT para equiparar la consulta con el mapeo de agregación sobre una tabla única. El formato de las consultas es SQL es el siguiente:
    - INSERT INTO <$tabla_1$> VALUES (<$valor_{1,1}$>, <$valor_{1,2}$>, … <$valor_{1,n}$>);
    - INSERT INTO <$tabla_2$> VALUES (<$valor_{2,1,1}$>, <$valor_{2,1,2}$>, … <$valor_{2,1,m}$>);
    - INSERT INTO <$tabla_2$> VALUES (<$valor_{2,2,1}$>, <$valor_{2,2,2}$>, … <$valor_{2,2,m}$>)

La tabla 1.1.6 muestra las consultas Insert realizadas sobre los distintos sistemas gestores de bases de datos.

MS Access	Oracle	MySQL	Postgres
INSERT INTO Tipo_Direccion VALUES ('123456789','Mayor','Pamplona','31115') INSERT INTO Tipo_Direccion VALUES ('987654321','Menor','Tudela','31000') INSERT INTO Cliente VALUES ('123456789','Juan','Martin Salas',1500,'123456789','987654321')	INSERT INTO Tipo_Direccion VALUES ('123456789','Mayor','Pamplona','31115') INSERT INTO Tipo_Direccion VALUES ('987654321','Menor','Tudela','31000') INSERT INTO Cliente VALUES ('123456789','Juan','Martin Salas',1500,'123456789','987654321')	INSERT INTO Tipo_Direccion VALUES ('123456789','Mayor','Pamplona','31115') INSERT INTO Tipo_Direccion VALUES ('987654321','Menor','Tudela','31000') INSERT INTO Cliente VALUES ('123456789','Juan','Martin Salas',1500,'123456789','987654321')	INSERT INTO Tipo_Direccion VALUES ('123456789','Mayor','Pamplona','31115') INSERT INTO Tipo_Direccion VALUES ('987654321','Menor','Tudela','31000') INSERT INTO Cliente VALUES ('123456789','Juan','Martin Salas',1500,'123456789','987654321')

Tabla 1.1.6 Consultas Insert por clave extranjera

UPDATE

- Update <table> set <field>=<value>
  - Se va a actualizar el valor de un campo en todos los registros de una tabla, pero al trabajar sobre dos tablas la petición UPDATE no tiene el mismo formato en los diferentes sistemas gestores de bases de datos, debido a que Oracle y Postgres, en este caso no soportan la cláusula INNER JOIN.
    - MS Access:
      - UPDATE <$tabla_2$> INNER JOIN <$tabla_1$> ON <$tabla_2$>.<$campo_v$> = <$tabla_1$>.<$campo_y$> SET <$tabla_1$>.<$campo_x$> = <*valor*>
    - Oracle: se ha tenido que recurrir a una transformación de la consulta de manera que sin contener INNER JOIN el resultado sea el mismo. Para ello, se ha empleado una subconsulta con la clausula ANY.
      - UPDATE <$tabla_1$> SET <$tabla_1$>.<$campo_x$> = <*valor*> WHERE <$tabla_1$>.<$campo_y$> = ANY (SELECT <$campo_v$> FROM <$tabla_2$>)
    - MySQL: semejante a MS Access.
      - UPDATE <$tabla_2$> INNER JOIN <$tabla_1$> ON <$tabla_2$>.<$campo_v$> = <$tabla_1$>.<$campo_y$> SET <$tabla_1$>.<$campo_x$> = <*valor*>
    - Postgres: no soporta la cláusula INNER JOIN. Se ha incluido la condición de que las claves principales de las tablas implicadas sean las mismas.

- UPDATE <*tabla*$_1$> SET <*campo*$_x$> = <*valor*> FROM <*tabla*$_2$> WHERE <*tabla*$_2$>.<*campo*$_v$> = <*tabla*$_1$>.<*campo*$_y$>

- Update <tabla> set <campo>=<valor> where <condición>
  - Se trata de actualizar el valor de un campo de los registros de una tabla que cumplan un criterio. Este criterio lo debe cumplir un campo que se encuentra en la otra tabla. Al igual que en el caso anterior no se ha podido realizar la consulta de manera semejante en todos los sistemas gestores, debido a que algunos gestores no soportan INNER JOIN en el UPDATE.
    - MS Access:
      - UPDATE <*tabla*$_2$> INNER JOIN <*tabla*$_1$> ON <*tabla*$_2$>.<*campo*$_v$> = <*tabla*$_1$>.<*campo*$_y$> SET <*tabla*$_1$>.<*campo*$_x$> = <*valor*$_1$> WHERE <*tabla*$_2$>.<*campo*$_m$> = <*valor*$_2$>
    - Oracle: no soporta INNER JOIN de modo que se ha vuelto a recurrir al empleo de una subconsulta con la cláusula ANY siendo esta subconsulta la que lleva implícita el criterio de actualización.
      - UPDATE <*tabla*$_1$> SET <*tabla*$_1$>.<*campo*$_x$> = <*valor*> WHERE <*tabla*$_1$>.<*campo*$_y$> = ANY (SELECT <*campo*$_v$> FROM <*tabla*$_2$> WHERE <campo$_m$>=<valor$_2$>
    - MySQL:
      - UPDATE <*tabla*$_2$> INNER JOIN <*tabla*$_1$> ON <*tabla*$_2$>.<*campo*$_v$> = <*tabla*$_1$>.<*campo*$_y$> SET <*tabla*$_1$>.<*campo*$_x$> = <*valor*$_1$> WHERE <*tabla*$_2$>.<*campo*$_m$> = <*valor*$_2$>
    - Postgres: el formato de la consulta es similar a la del caso anterior, pero dado que se necesita una condición la hemos añadido empleando AND, de modo que se incluyen dos condiciones; la primera, en la igualdad de las claves principales y la segunda, en la condición inicial del UPDATE.
      - UPDATE <*tabla*$_1$> SET <*campo*$_x$> = <*valor*> FROM <*tabla*$_2$> WHERE <*tabla*$_2$>.<*campo*$_v$> = <*tabla*$_1$>.<*campo*$_y$> AND <*tabla2*>.<*campo*$_m$> = <*valor*$_2$>
- Update <tablas> set <campos> = <valores> where <condición>
  - Lo que se pretende con esta consulta es la actualización de dos campos que se encuentran en diferentes tablas y el criterio de actualización se encuentra en una de las tablas.
    - MS Access: dado que soporta la cláusula INNER JOIN se ha hecho uso de ella.
      - UPDATE <*tabla*$_2$> INNER JOIN <*tabla*$_1$> ON <*tabla*$_2$>.<*campo*$_v$> = <*tabla*$_1$>.<*campo*$_y$> SET <*tabla*$_1$>.<*campo*$_x$> = <*valor*$_1$>, <*tabla*$_2$>.<*campo*$_x$> = <*valor*$_2$> WHERE <*tabla*$_1$>.<*campo*$_m$> < <*valor*$_3$>

- Oracle: se ha tendido que optar por la combinación de dos consultas de actualización para llevar a cabo actualizaciones en las dos tablas. En la tabla que no contiene el criterio, se ha empleado una subconsulta con ANY conteniendo la condición de actualización, y en la tabla que contiene el criterio se ha empleado un UPDATE con el criterio de actualización.
  - UPDATE <$tabla_1$> SET <$tabla_1$>.<$campo_x$> = <*valor*> WHERE <$tabla_1$>.<$campo_y$> = ANY (SELECT <$campo_m$> FROM <$tabla_1$> WHERE <$campo_m$> < <$valor_2$>)
  - UPDATE <$tabla_1$> SET <$campo_x$> = <$valor_1$> WHERE <$campo_m$> < <$valor_3$>
- MySQL: al admitir INNER JOIN la petición resulta identica a la empleada en MS Access.
  - UPDATE <$tabla_2$> INNER JOIN <$tabla_1$> ON <$tabla_2$>.<$campo_v$> = <$tabla_1$>.<$campo_y$> SET <$tabla_1$>.<$campo_x$> = <$valor_1$>, <$tabla_2$>.<$campo_x$> = <$valor_2$> WHERE <$tabla_1$>.<$campo_m$> < <$valor_3$>
- Postgres: se ha incluido la actualización de los dos campos en una sóla consulta, a pesar de que dichos campos pertenecen a tablas diferentes. Para ello se ha empleado el siguiente formato.
  - UPDATE <$tabla_1$> SET <$campo_y$> = <*valor*>, <$tabla_2$>.<$campo_x$> WHERE <$tabla_1$>.<$campo_m$> < <$valor_3$>

La tabla 1.1.7 muestra las consultas Update realizadas sobre los distintos sistemas gestores de bases de datos.

MS Access	Oracle	MySQL	Postgres
UPDATE Tipo_Direccion INNER JOIN Cliente ON Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega SET Cliente.Nombre = 'Fuera Zona'	UPDATE Cliente SET Nombre='Fuera Zona' WHERE Cliente.Dire_Entrega = ANY (SELECT Id_Direccion FROM Tipo_Direccion)	UPDATE Tipo_Direccion INNER JOIN Cliente ON Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega SET Cliente.Nombre = 'Fuera Zona'	UPDATE Cliente SET Nombre='Fuera Zona' FROM Tipo_Direccion WHERE Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega
UPDATE Tipo_Direccion INNER JOIN Cliente ON Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega SET Cliente.Nombre = 'Fuera Zona' WHERE (((Tipo_Direccion.Calle) ='Mayor'))	UPDATE Cliente SET Nombre='Fuera Zona' WHERE Dire_entrega= ANY (SELECT Id_Direccion FROM Tipo_Direccion WHERE calle='Mayor')	UPDATE Tipo_Direccion INNER JOIN Cliente ON Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega SET Cliente.Nombre = 'Fuera Zona' WHERE (((Tipo_Direccion.Calle) ='Mayor'))	UPDATE Cliente SET Nombre = 'Fuera Zona' FROM Tipo_Direccion WHERE Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega AND Tipo_Direccion.Calle=' Mayor'
UPDATE Tipo_Direccion INNER JOIN Cliente ON	UPDATE Tipo_Direccion SET Calle='Mayor' WHERE	UPDATE Tipo_Direccion INNER JOIN Cliente ON	UPDATE Cliente SET Nombre = 'Anulado', Tipo_Direccion.Calle =

Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega SET Cliente.Nombre = 'Anulado', Tipo_Direccion.Calle = 'Mayor' WHERE (((Cliente.Total_Facturado)<200))	Id_Direccion=ANY (SELECT Dire_Entrega FROM Cliente WHERE Total_Facturado<200) UPDATE Cliente SET Nombre='Fuera Zona' WHERE Total_Facturado<200	Tipo_Direccion.Id_Direccion = Cliente.Dire_Entrega SET Cliente.Nombre = 'Anulado', Tipo_Direccion.Calle = 'Mayor' WHERE (((Cliente.Total_Facturado)<200))	'Mayor' WHERE (((Cliente.Total_Facturado)<200))

Tabla 1.1.7 Consultas Update por clave extranjera

DELETE

- Delete * from <tabla>
  - En el caso de agregación en tabla única el '*delete * from <tabla>*' eliminaba todos los registros de la tabla, para hacer equivalente esa consulta con la agregación por tabla extranjera se va a hacer lo mismo. Para ello se elimina la tabla que contiene la clave principal (Id_Direccion) y debido borrado en cascada y a la integridad referencial se eliminan todos los registros de la segunda tabla, de modo que el resultado final es semejante. La consulta es la misma para todos los sistemas gestores de bases de datos.

    - DELETE FROM <*tabla*>

- Delete * where <condición>
  - La condición se evalúa en una de las tablas y al tener que realizar el borrado equivalente sobre agregación en tabla única, se deben eliminar de las dos tablas los registros que deben estar en el mapeo de agregación de tabla única. Para ello se ha construido una consulta compatible con todos los sistemas gestores basada en subconsultas con cláusulas ANY y una condición OR; La consulta resultante tiene el siguiente formato:

    - DELETE FROM <$tabla_2$> WHERE (<$tabla_2$>.<$campo_v$> = ANY (SELECT <$tabla_1$>.<$campo_y$> FROM <$tabla_1$> WHERE <$campo_x$> < <*valor*>) OR <$tabla_2$>.<$campo_v$> = ANY (SELECT <$tabla_1$>.<$campo_n$> FROM <$tabla_1$> WHERE <$campo_x$> < <*valor*>)

La tabla 1.1.8 muestra las consultas Update realizadas sobre los distintos sistemas gestores de bases de datos

MS Access	Oracle	MySQL	Postgres
DELETE FROM Tipo_direccion WHERE (Tipo_Direccion.Id_Direccion=ANY (SELECT Cliente.dire_entrega from cliente where total_facturado<1000) Or Tipo_Direccion.Id_Direccion=ANY(SELECT Cliente.Dire_Factura FROM cliente where total_facturado<1000))	DELETE FROM Tipo_direccion WHERE (Tipo_Direccion.Id_Direccion=ANY (SELECT Cliente.dire_entrega from cliente where total_facturado<1000) Or Tipo_Direccion.Id_Direccion=ANY(SELECT Cliente.Dire_Factura FROM cliente where total_facturado<1000))	DELETE FROM Tipo_direccion WHERE (Tipo_Direccion.Id_Direccion=ANY (SELECT Cliente.dire_entrega from cliente where total_facturado<1000) Or Tipo_Direccion.Id_Direccion=ANY(SELECT Cliente.Dire_Factura FROM cliente where total_facturado<1000))	DELETE FROM Tipo_direccion WHERE (Tipo_Direccion.Id_Direccion=ANY (SELECT Cliente.dire_entrega from cliente where total_facturado<1000) Or Tipo_Direccion.Id_Direccion=ANY(SELECT Cliente.Dire_Factura FROM cliente where total_facturado<1000))

Tabla 1.1.8 Consultas Delete por clave extranjera

1.2 ASOCIACIÓN

La figura 1.2.1 muestra el diagrama de clases de la asociación.

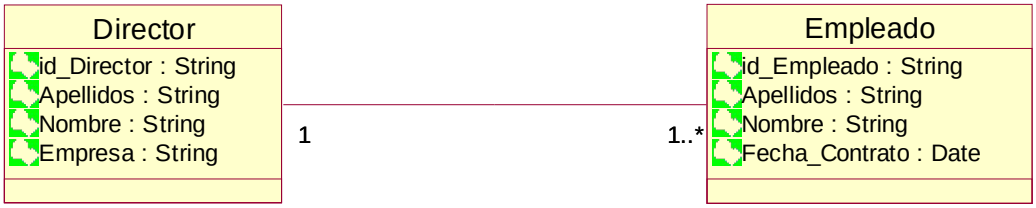


Figura 1.2.1 Diagramas de clases de la asociación

La asociación tiene dos formas de mapeo, mapeo por clave extranjera y mapeo por tabla de asociación. Se muestran dichos mapeos y las consultas que se han realizado sobre ellos.

- *Mapeo por clave extranjera*: la figura 1.2.2 muestra la tabla relacional consecuencia del mapeo en tabla única de la agregación.

- Clave extranjera:

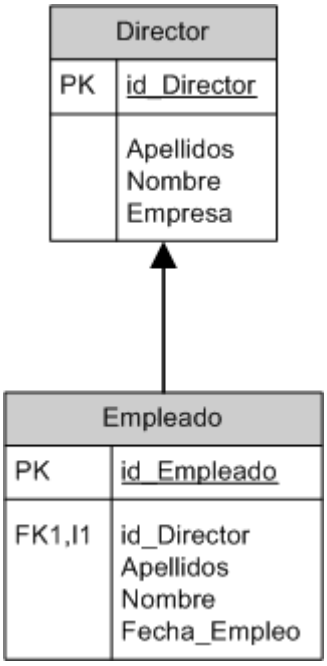


Figura 1.2.2 Mapeo por clave extranjera

SELECT

- Select * from <tabla>
  - Para conseguir que muestre todos los registros, se realiza un SELECT sobre dos tablas utilizando INNER JOIN. Todos los sistemas gestores han admitido el mismo formato:
    - SELECT <*tabla*$_1$>.*, <*tabla*$_2$>.* FROM <*tabla*$_1$> INNER JOIN <*tabla*$_2$> ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_2$>.<*campo*$_1$>

- Select <campos> from <tabla>
  - Se ha realizado una consulta SELECT que muestra varios campos de una de las tablas. Dada la compatibilidad de las consultas SELECT el formato es el mismo para todos los sistemas.
    - SELECT <*campo*$_1$>, <*campo*$_2$>, … , <*campo*$_x$> FROM <*tabla*$_2$>

- Select * from <tabla> where <condición>
  - La consulta va a mostrar los campos de las dos tablas utilizando INNER JOIN con un criterio aplicado a una de ellas. Consulta admitida por todos los sistemas gestores. Pese a que el formato de las fechas es diferente entre ellos, se ha utilizado el empleado por MS Access ya que el programa de medida de rendimiento adapta el formato de las fechas al sistema gestor que ataca.

- SELECT <*tabla*$_1$>.*, <*tabla*$_2$>.* FROM <*tabla*$_1$> INNER JOIN <*tabla*$_2$> ON <*tabla*$_2$>.<*campo*$_1$> = <*tabla*$_1$>.<*campo*$_1$> WHERE <*tabla*$_2$>.<*campo*$_x$> > <*valor*>

o Select <campos> from <tabla> where <condición>

- Muestra los valores de determinados campos que se encuentan en ambas tablas cumpliendo un criterio que se aplica a una de ellas. Todos los sistemas gestores han admitido el uso de INNER JOIN y se ha obtenido una consulta SQL admitida por todos. Al igual que en el caso anterior la fecha tiene formato de MS Access dado que el programa de medida traslada a la empleada por el sistema gestor sobre el que se está ejecutándose.

  - SELECT <*tabla*$_1$>.<*campo*$_m$>,, …, <*tabla*$_1$>.<*campo*$_v$>, <*tabla*$_2$>.<*campo*$_h$>, …, <*tabla*$_2$>.<*campo*$_j$> FROM <*tabla*$_1$> INNER JOIN <*tabla*$_2$> ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_2$>.<*campo*$_1$> WHERE <*tabla*$_2$>.<*campo*$_g$> > <*valor*>

La tabla 1.2.1 muestra las consultas Select realizadas sobre los distintos sistemas gestores de bases de datos

MS Access	Oracle	MySQL	Postgres
SELECT Director.*, Empleado.* FROM Director INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director	SELECT Director.*, Empleado.* FROM Director INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director	SELECT Director.*, Empleado.* FROM Director INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director	SELECT Director.*, Empleado.* FROM Director INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director
SELECT Director.Apellidos, Director.Nombre, Director.Empresa, Empleado.Fecha_Empleo, Empleado.Nombre FROM Director INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director	SELECT Director.Apellidos, Director.Nombre, Director.Empresa, Empleado.Fecha_Empleo, Empleado.Nombre FROM Director INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director	SELECT Director.Apellidos, Director.Nombre, Director.Empresa, Empleado.Fecha_Empleo, Empleado.Nombre FROM Director INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director	SELECT Director.Apellidos, Director.Nombre, Director.Empresa, Empleado.Fecha_Empleo, Empleado.Nombre FROM Director INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director
SELECT Director.*, Empleado.* FROM Director INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director WHERE (((Empleado.Fecha_Empleo)>#1/1/2004#))	SELECT Director.*, Empleado.* FROM Director INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director WHERE (((Empleado.Fecha_Empleo)>#1/1/2004#))	SELECT Director.*, Empleado.* FROM Director INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director WHERE (((Empleado.Fecha_Empleo)>#1/1/2004#))	SELECT Director.*, Empleado.* FROM Director INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director WHERE (((Empleado.Fecha_Empleo)>#1/1/2004#))
SELECT Director.Apellidos, Director.Nombre, Director.Empresa, Empleado.Fecha_Empleo FROM Director	SELECT Director.Apellidos, Director.Nombre, Director.Empresa, Empleado.Fecha_Empleo FROM Director	SELECT Director.Apellidos, Director.Nombre, Director.Empresa, Empleado.Fecha_Empleo FROM Director	SELECT Director.Apellidos, Director.Nombre, Director.Empresa, Empleado.Fecha_Empleo FROM Director

INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director WHERE (((Empleado.Fecha_Empleo)>#1/1/2004#))	INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director WHERE (((Empleado.Fecha_Empleo)>#1/1/2004#))	INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director WHERE (((Empleado.Fecha_Empleo)>#1/1/2004#))	INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director WHERE (((Empleado.Fecha_Empleo)>#1/1/2004#))

Tabla 1.2.1 Consultas Select clave extranjera

INSERT

- Insert into <tabla> values (<valores>)
  - Se insertan dos registros con todos sus campos en ambas tablas. Para ello, se emplea dos consultas INSERT que son admitidas por todos los sistemas gestores.

    - INSERT INTO <$tabla_1$> VALUES (<$valor_1$>, <$valor_2$>, … , <$valor_n$>)
    - INSERT INTO <$tabla_2$> VALUES (<$valor_1$>, <$valor_2$>, … , <$valor_m$>)

La tabla 1.2.2 muestra las consultas Insert realizadas sobre los distintos sistemas gestores de bases de datos

MS Access	Oracle	MySQL	Postgres
INSERT INTO Director VALUES (123456789,'Cortes Salas','Fernando','Tornillos Cortes') $%& INSERT INTO Empleado VALUES ('123456789',123456789,'Morales Martin','Francisco',#25/7/2003#)	INSERT INTO Director VALUES (123456789,'Cortes Salas','Fernando','Tornillos Cortes') $%& INSERT INTO Empleado VALUES ('123456789',123456789,'Morales Martin','Francisco',#25/7/2003#)	INSERT INTO Director VALUES (123456789,'Cortes Salas','Fernando','Tornillos Cortes') $%& INSERT INTO Empleado VALUES ('123456789',123456789,'Morales Martin','Francisco',#25/7/2003#)	INSERT INTO Director VALUES (123456789,'Cortes Salas','Fernando','Tornillos Cortes') $%& INSERT INTO Empleado VALUES ('123456789',123456789,'Morales Martin','Francisco',#25/7/2003#)

Tabla 1.2.2 Consultas Insert clave extranjera

UPDATE

- Update <tabla> set <campo> = <valor>
  - La consulta se limita a actualizar un campo concreto de todos los registros de una de las tablas. Al tratarse de un UPDATE muy sencillo todos los sistemas gestores lo han procesado.

    - UPDATE <$tabla_1$> SET <$campo_x$> = <valor>

- Update <tabla> set <campo>=<valor> where <condición>

- Actualiza un campo de los registros de una de las tablas cuando se cumple una determinada condición en la otra tabla. Dado que no es una consulta sencilla puesto que se trabaja sobre las dos tablas, una con la actualización y otra con el criterio de actualización, no se ha conseguido un formato de consulta que sirva para todos los sistemas gestores.
  - MS Access:
    - Se ha utilizado la cláusula INNER JOIN y se ha aplicado el UPDATE con la condición sobre la tabla resultante.
      - UPDATE <*tabla*$_1$> INNER JOIN <*tabla*$_2$> ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_2$>.<*campo*$_x$> SET <*tabla*$_2$>.<*campo*$_v$> = <*valor*$_1$> WHERE <*tabla*$_1$>.<*campo*$_u$> = <*valor*$_2$>
  - Oracle:
    - Se ha empleado UPDATE con la condición de que el campo con clave extranjera (Id_Director) se encontrase en una subconsulta que contiene la condición inicial.
      - UPDATE <*tabla*$_2$> SET <*campo*$_v$> = <*valor*$_1$> WHERE <*campo*$_x$> = ANY (SELECT <*campo*$_1$> FROM <*tabla*$_1$> WHERE <*campo*$_u$> = <*valor*$_2$>
  - MySQL:
    - Al igual que MS Access, este sistema gestor ha ejecutado la cláusula INNER JOIN.
      - UPDATE <*tabla*$_1$> INNER JOIN <*tabla*$_2$> ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_2$>.<*campo*$_x$> SET <*tabla*$_2$>.<*campo*$_v$> = <*valor*$_1$> WHERE <*tabla*$_1$>.<*campo*$_u$> = <*valor*$_2$>
  - Postgres:
    - Se ha utilizado una consulta UPDATE que incluye dos condiciones enlazadas mediante un AND. Una de las condiciones es que el campo clave extranjera (Id_Director) de una de las tablas sea igual que la clave principal de la otra tabla y la otra condición la condición inicial.
      - UPDATE <*tabla*$_2$> SET <*tabla*$_2$>.<*campo*$_v$> = <*valor*$_1$> FROM <tabla1> WHERE <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_2$>.<*campo*$_x$> AND <*tabla*$_1$>.<*campo*$_u$> = <*valor*$_2$>

- Update <tabla> set <campos> = <valores> where <condición>
  - Con esta consulta se pretende actualizar dos campos que pertenecen a tablas diferentes siguiendo un criterio sobre una de ellas. Al igual que para el caso anterior no se ha conseguido un formato unificado para todos los sistemas gestores.
    - MS Access:
      - Admite la cláusula INNER JOIN de modo que el formato de la petición resultante es sencillo, basta con indicarle los campos que se desean actualizar.

        - UPDATE <*tabla*$_1$> INNER JOIN <*tabla*$_2$> ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_2$>.<*campo*$_x$> SET <*tabla*$_1$>.<*campo*$_g$> = <*valor*$_3$>, <*tabla*$_2$>.<*campo*$_v$> = <*valor*$_4$> WHERE <*tabla*$_2$>.<*campo*$_h$> = <*valor*$_5$>

- Oracle:
    - Al tener que actualizar campos en las dos tablas y no admitir la clausula INNER JOIN se ha tenido que recurrir al empleo de dos consultas UPDATE. La primera realiza la actualización sobre la tabla que no contiene el campo sobre el que se aplica el criterio y por ello se emplea una condición enlazada a una subconsulta empleando ANY. La segunda consulta es un UPDATE sobre la tabla que contiene el campo sobre la que se aplica el criterio inicial de modo que resulta una consulta UPDATE con una condición.
        - UPDATE <*tabla*$_1$> SET <*tabla*$_1$>.<*campo*$_g$> = <*valor*$_3$> WHERE <tabla1>.<campo1> = ANY (SELECT <campo2> FROM <*tabla*$_2$> WHERE <*tabla*$_2$>.<*campo*$_h$> = <*valor*$_5$>)
        - UPDATE <tabla2> SET <*tabla*$_2$>.<*campo*$_v$> = <*valor*$_4$> WHERE <*tabla*$_2$>.<*campo*$_h$> = <*valor*$_5$>
- MySQL:
    - Al igual que MS Access admite INNER JOIN, de modo que el formato de la consulta resultante es el mismo que para Access.
        - UPDATE <*tabla*$_1$> INNER JOIN <*tabla*$_2$> ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_2$>.<*campo*$_x$> SET <*tabla*$_1$>.<*campo*$_g$> = <*valor*$_3$>, <*tabla*$_2$>.<*campo*$_v$> = <*valor*$_4$> WHERE <*tabla*$_2$>.<*campo*$_h$> = <*valor*$_5$>
- Postgres:
    - No admite INNER JOIN de modo que al igual que ha ocurrido para Oracle se han tenido que realizar dos consultas UPDATE. La primera consulta para actualizar la tabla sobre la que no se impone la condición inicial (Director). para ello se emplea una doble condición enlazada por la cláusula AND. la primera que su clave principal sea igual que la clave extranjera de la otra tabla y la segunda es la condición que consideramos inicial. El segundo UPDATE se aplica sobre la tabla a la que hemos aplicado la condición inicial de modo que resulta una petición UPDATE sencilla con una única condición.
        - UPDATE <*tabla*$_1$> SET <*tabla*$_1$>.<*campo*$_g$> = <*valor*$_3$> FROM <*tabla*$_2$> WHERE <tabla1>.<campo1> = <*tabla*$_2$>.<*campo*$_x$> AND <*tabla*$_2$>.<*campo*$_h$> = <*valor*$_5$>

- UPDATE <tabla2> SET <*tabla*$_2$>.<*campo*$_v$> = <*valor*$_4$> WHERE <*tabla*$_2$>.<*campo*$_h$> = <*valor*$_5$>

La tabla 1.2.3 muestra las consultas Update realizadas sobre los distintos sistemas gestores de bases de datos.

MS Access	Oracle	MySQL	Postgres
UPDATE Director SET Nombre = 'Pedro'	UPDATE Director SET Nombre = 'Pedro'	UPDATE Director SET Nombre = 'Pedro'	UPDATE Director SET Nombre = 'Pedro'
UPDATE Director INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director SET Empleado.Apellidos = 'Despedido' WHERE Director.Empresa='Telefonica'	UPDATE Empleado SET Apellidos='Martinez' WHERE id_Director= ANY (SELECT id_director FROM Director WHERE Empresa='Telefonica')	UPDATE Director INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director SET Empleado.Apellidos = 'Despedido' WHERE Director.Empresa='Telefonica'	UPDATE Empleado SET Apellidos = 'Martinez' FROM Director WHERE Director.id_Director=Empleado.id_Director AND Director.Empresa='Telefonica'
UPDATE Director INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director SET Director.Nombre = 'Pedro', Empleado.Apellidos = 'Gonzalez' WHERE (((Empleado.Fecha_Empleo)>#1/1/2005#))	UPDATE Director SET Nombre = 'Pedro' WHERE Director.id_Director= ANY (SELECT id_Director FROM Empleado WHERE Fecha_Empleo>#1/1/2005#) UPDATE Empleado SET Apellidos= 'Gonzalez' WHERE Empleado.Fecha_Empleo>#1/1/2005#	UPDATE Director INNER JOIN Empleado ON Director.id_Director = Empleado.id_Director SET Director.Nombre = 'Pedro', Empleado.Apellidos = 'Gonzalez' WHERE (((Empleado.Fecha_Empleo)>#1/1/2005#))	UPDATE Director SET Nombre = 'Pedro' FROM Empleado WHERE Director.id_Director=Empleado.id_Director AND Empleado.Fecha_Empleo>#1/1/2005# UPDATE Empleado SET Apellidos = 'Gonzalez' FROM Director WHERE Fecha_Empleo>#1/1/2005#

Tabla 1.2.3 Consultas Update clave extranjera

DELETE

- Delete * from <tabla>
  - Deseamos eliminar los registros de todas las tablas, para ello eliminamos la tabla en la que se encuentra la clave extranjera y mediante un borrado en cascada se eliminarán (manteniendo la integridad referencial) todos los registros de la otra. La sentencia es admitida por todos los sistemas gestores que no se escriba el '*'. El formato de la sentencia es el siguiente.
    - DELETE FROM <*tabla*$_1$>
- Delete where <condición>

- En este caso se desea eliminar todos aquellos registros de una de las tablas que cumplan un cierto criterio, y mediante un borrado en cascada para mantener la integridad referencial, se eliminan los registros de la tabla que aloja la clave principal con las que está relacionado. La consulta es igual para todos los gestores.

    - DELETE FROM <$tabla_1$> WHERE <$tabla_1$>.<$campo_x$> = <$valor_1$>

La tabla 1.2.4 muestra las consultas Delete realizadas sobre los distintos sistemas gestores de bases de datos

MS Access	Oracle	MySQL	Postgres
DELETE FROM Director	DELETE FROM Director	DELETE FROM Director	DELETE FROM Director
DELETE FROM Director WHERE (((Director.Empresa)='Telefonica'))	DELETE FROM Director WHERE (((Director.Empresa)='Telefonica'))	DELETE FROM Director WHERE (((Director.Empresa)='Telefonica'))	DELETE FROM Director WHERE (((Director.Empresa)='Telefonica'))

Tabla 1.2.4 Consultas Delete clave extranjera

- *Mapeo por tabla de asociación*: la figura 1.2.3 muestra las tablas relacionales y sus relaciones del mapeo por tabla de asociación.

Tabla de asociación

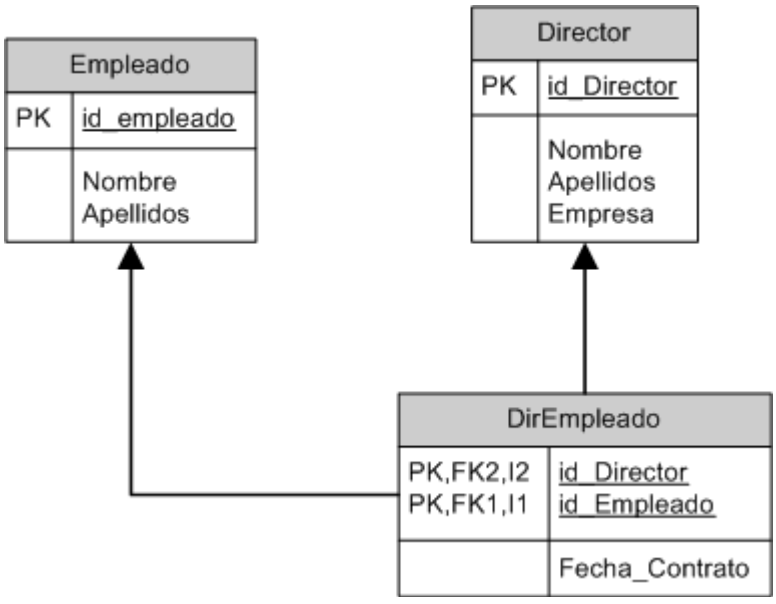


Figura 1.2.3 Mapeo por tabla de asociación

SELECT

- Select * from <tabla>
  - Se muestran todos los campos de las tres tablas relacionadas. Para ello realizamos una consulta SELECT con dos cláusulas INNER JOIN. Todos los sistemas gestores admiten INNER JOIN con el mismo formato en las consultas SELECT.
    - SELECT <*tabla*$_1$>.*, <*tabla*$_2$>.*, <*tabla*$_3$>.* FROM <*tabla*$_1$> INNER JOIN (<*tabla*$_2$> INNER JOIN <*tabla*$_3$> ON <*tabla*$_2$>.<*campo*$_1$> = <*tabla*$_3$>.<*campo*$_1$>) ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_3$>.<*campo*$_1$>

- Select <campos> from <tabla>
  - Se muestran determinados campos de las tres tablas. Para ello se realiza una consulta similar a la anterior, pero en este caso, en lugar de mostrar todos los campos de las tablas empleando <tabla>.*, se seleccionan los campos que vamos a mostrar utilizando <tabla>.<campo>. Al igual que en el caso anterior la consulta tiene un formato totalmente compatible con todos los sistemas gestores.
    - SELECT <*tabla*$_1$>.<*campo*$_u$>, <*tabla*$_2$>.<*campo*$_v$>, <*tabla*$_3$>.<*campo*$_h$> FROM <*tabla*$_1$> INNER JOIN (<*tabla*$_2$> INNER JOIN <*tabla*$_3$> ON <*tabla*$_2$>.<*campo*$_1$> = <*tabla*$_3$>.<*campo*$_1$>) ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_3$>.<*campo*$_1$>

- Select * from <tabla> where <condición>
  - Muestra todos los campos de las tablas con un criterio sobre una de las tablas y al estar éstas interrelacionadas, afecta a todas ellas. El formato de la consulta es similar al de la empleada en 'select * from <tabla>', pero añadiendo un criterio mediante WHERE. Todos los sistemas gestores admiten la misma consulta.
    - SELECT <*tabla*$_1$>.*, <*tabla*$_2$>.*, <*tabla*$_3$>.* FROM <*tabla*$_1$> INNER JOIN (<*tabla*$_2$> INNER JOIN <*tabla*$_3$> ON <*tabla*$_2$>.<*campo*$_1$> = <*tabla*$_3$>.<*campo*$_1$>) ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_3$>.<*campo*$_1$> WHERE <*tabla*$_3$>.<*campo*$_x$> > <*valor*>

- Select <campos> from <tabla> where <condición>
  - Se trata de una consulta similar a la anterior, pero en lugar de mostrar todos los campos, se van a mostrar algunos de ellos. Para ello, como ya se ha visto en casos anteriores, se sustituye

<tabla>.* por <tabla>.<campo>. Esta consulta es compatible con todos los sistemas gestores.

- SELECT *<tabla$_1$>.<campo$_u$>*, *<tabla$_2$>.<campo$_v$>*, *<tabla$_3$>.<campo$_h$>* FROM *<tabla$_1$>* INNER JOIN *(<tabla$_2$>* INNER JOIN *<tabla$_3$>* ON *<tabla$_2$>.<campo$_1$>* = *<tabla$_3$>.<campo$_1$>)* ON *<tabla$_1$>.<campo$_1$>* = *<tabla$_3$>.<campo$_1$>* WHERE *<tabla$_3$>.<campo$_x$>* > *<valor>*

La tabla 1.2.5 muestra las consultas Select realizadas sobre los distintos sistemas gestores de bases de datos

MS Access	Oracle	MySQL	Postgres
SELECT Director.*, DirEmpleado.*, Empleado.* FROM Empleado INNER JOIN (Director INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado = DirEmpleado.id_Empleado	SELECT Director.*, DirEmpleado.*, Empleado.* FROM Empleado INNER JOIN (Director INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado = DirEmpleado.id_Empleado	SELECT Director.*, DirEmpleado.*, Empleado.* FROM Empleado INNER JOIN (Director INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado = DirEmpleado.id_Empleado	SELECT Director.*, DirEmpleado.*, Empleado.* FROM Empleado INNER JOIN (Director INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado = DirEmpleado.id_Empleado
SELECT Director.Nombre, Director.Apellidos, Director.Empresa, DirEmpleado.Fecha_Contrato, Empleado.Nombre FROM Empleado INNER JOIN (Director INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado = DirEmpleado.id_Empleado	SELECT Director.Nombre, Director.Apellidos, Director.Empresa, DirEmpleado.Fecha_Contrato, Empleado.Nombre FROM Empleado INNER JOIN (Director INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado = DirEmpleado.id_Empleado	SELECT Director.Nombre, Director.Apellidos, Director.Empresa, DirEmpleado.Fecha_Contrato, Empleado.Nombre FROM Empleado INNER JOIN (Director INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado = DirEmpleado.id_Empleado	SELECT Director.Nombre, Director.Apellidos, Director.Empresa, DirEmpleado.Fecha_Contrato, Empleado.Nombre FROM Empleado INNER JOIN (Director INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado = DirEmpleado.id_Empleado
SELECT Director.*, DirEmpleado.*, Empleado.* FROM Empleado INNER JOIN (Director INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado	SELECT Director.*, DirEmpleado.*, Empleado.* FROM Empleado INNER JOIN (Director INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado	SELECT Director.*, DirEmpleado.*, Empleado.* FROM Empleado INNER JOIN (Director INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado	SELECT Director.*, DirEmpleado.*, Empleado.* FROM Empleado INNER JOIN (Director INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado

= DirEmpleado.id_Empleado WHERE (((DirEmpleado.Fecha_Contrato)>#1/1/2004#))	= DirEmpleado.id_Empleado WHERE (((DirEmpleado.Fecha_Contrato)>#1/1/2004#))	= DirEmpleado.id_Empleado WHERE (((DirEmpleado.Fecha_Contrato)>#1/1/2004#))	= DirEmpleado.id_Empleado WHERE (((DirEmpleado.Fecha_Contrato)>#1/1/2004#))
SELECT Director.Nombre, Director.Apellidos, Director.Empresa, DirEmpleado.Fecha_Contrato, Empleado.Nombre FROM Empleado INNER JOIN (Director INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado = DirEmpleado.id_Empleado WHERE (((DirEmpleado.Fecha_Contrato)>#1/1/2004#))	SELECT Director.Nombre, Director.Apellidos, Director.Empresa, DirEmpleado.Fecha_Contrato, Empleado.Nombre FROM Empleado INNER JOIN (Director INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado = DirEmpleado.id_Empleado WHERE (((DirEmpleado.Fecha_Contrato)>#1/1/2004#))	SELECT Director.Nombre, Director.Apellidos, Director.Empresa, DirEmpleado.Fecha_Contrato, Empleado.Nombre FROM Empleado INNER JOIN (Director INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado = DirEmpleado.id_Empleado WHERE (((DirEmpleado.Fecha_Contrato)>#1/1/2004#))	SELECT Director.Nombre, Director.Apellidos, Director.Empresa, DirEmpleado.Fecha_Contrato, Empleado.Nombre FROM Empleado INNER JOIN (Director INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado = DirEmpleado.id_Empleado WHERE (((DirEmpleado.Fecha_Contrato)>#1/1/2004#))

Tabla 1.2.5 Consultas Select asociación tabla

INSERT

- Insert into <tabla> values <valores>
  - Para que esta consulta resulte equivalente a la inserción en un mapeo de asociación con clave extranjera, se han de insertar tres registros en la base de datos, uno en cada tabla. Para ello se realizan tres consultas INSERT consecutivas. La sentencia es compatible en todos los sistemas gestores.

    - INSERT INTO <$tabla_1$> VALUES (<$valor_1$>, <$valor_2$>, … , <$valor_n$>)
    - INSERT INTO <$tabla_2$> VALUES (<$valor_1$>, <$valor_2$>, … , <$valor_m$>)
    - INSERT INTO <$tabla_3$> VALUES (<$valor_1$>, <$valor_2$>, … , <$valor_w$>)

La tabla 1.2.6 muestra las consultas Insert realizadas sobre los distintos sistemas gestores de bases de datos

MS Access	Oracle	MySQL	Postgres
INSERT INTO Director VALUES (123456789,'Fernando','	INSERT INTO Director VALUES (123456789,'Fernando','	INSERT INTO Director VALUES (123456789,'Fernando','	INSERT INTO Director VALUES (123456789,'Fernando','

Cortes Salas','Tornillos Cortes') INSERT INTO Empleado VALUES (123456789, 'Francisco','Morales Martin') INSERT INTO DirEmpleado VALUES (123456789,123456789, #25/7/2003#)	Cortes Salas','Tornillos Cortes') INSERT INTO Empleado VALUES (123456789, 'Francisco','Morales Martin') INSERT INTO DirEmpleado VALUES (123456789,123456789, #25/7/2003#)	Cortes Salas','Tornillos Cortes') INSERT INTO Empleado VALUES (123456789, 'Francisco','Morales Martin') INSERT INTO DirEmpleado VALUES (123456789,123456789, #25/7/2003#)	Cortes Salas','Tornillos Cortes') INSERT INTO Empleado VALUES (123456789, 'Francisco','Morales Martin') INSERT INTO DirEmpleado VALUES (123456789,123456789, #25/7/2003#)

Tabla 1.2.6 Consultas Insert asociación tabla

UPDATE

- Update <tabla> set <campo> = <valor>
  - La pconsulta se limita a actualizar un campo concreto de todos los registros de una de las tablas. La consulta es compatible con todos los sistemas gestores.
    - UPDATE <$tabla_1$> SET <$campo_x$> = <*valor*>
- Update <tabla> set <campo> = <valor> where <condición>
  - Actualiza un campo de los registros de una de las tablas cuando se cumple una determinada condición en la otra tabla. La consulta no es comptable con todos los sistemas gestores.
    - MS Access:
      - Se ha utilizado la cláusula INNER JOIN y se han aplicado el UPDATE con la condición sobre la tabla resultante.
        - UPDATE <$tabla_1$> INNER JOIN (<$tabla_2$> INNER JOIN <$tabla_3$> ON <$tabla_2$>.<$campo_1$> = <$tabla_3$>.<$campo_1$>) ON <$tabla_1$>.<$campo_1$> = <$tabla_3$>.<$campo_x$> SET <$tabla_1$>.<$campo_v$> = <$valor_1$> WHERE <$tabla_2$>.<$campo_u$> = <$valor_2$>
    - Oracle:
      - Como ya se ha podido ver en casos anteriores, Oracle no admite utilizar INNER JOIN en una consulta UPDATE. Por ello se ha vuelto a recurrir al uso de subconsultas. En este caso, al trabajar sobre tres tablas, se han utilizado dos subconsultas anidadas y en una de ellas se ha incluido el criterio de actualización.
        - UPDATE <$tabla_1$> SET <$campo_v$> = <$valor_1$> WHERE <$campo_1$> = ANY (SELECT <$campo_2$> FROM <$tabla_3$> WHERE <$campo_1$> = ANY (SELECT <$campo_1$> FROM <$tabla_2$> WHERE <$tabla_2$>.<$campo_u$> = <$valor_2$>

- MySQL:
  - Al igual que MS Access, este sistema gestor ha admitido la cláusula INNER JOIN de modo que el formato de la instrucción ha resultado igual que el utilizado en Access.
    - UPDATE <*tabla*$_1$> INNER JOIN (<*tabla*$_2$> INNER JOIN <*tabla*$_3$> ON <*tabla*$_2$>.<*campo*$_1$> = <*tabla*$_3$>.<*campo*$_1$>) ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_3$>.<*campo*$_x$> SET <*tabla*$_1$>.<*campo*$_v$> = <*valor*$_1$> WHERE <*tabla*$_2$>.<*campo*$_u$> = <*valor*$_2$>
- Postgres:
  - Se ha empleado una consuta UPDATE que incluye tres condiciones encadenadas mediante dos operadores lógicos AND. Dos de ellas aseguran la coincidencia de las claves extranjeras de una de las tablas con las claves principales de las otras tablas. La tercera condición contiene el criterio de actualización.
    - UPDATE <*tabla*$_1$> SET <*tabla*$_1$>.<*campo*$_v$> = <*valor*$_1$> FROM <*tabla*$_3$> WHERE <*tabla*$_3$>.<*campo*$_1$> = <*tabla*$_2$>.<*campo*$_1$> AND <*tabla*$_3$>.<*campo*$_2$> = <*tabla*$_1$>.<*campo*$_1$> AND <*tabla*$_2$>.<*campo*$_u$> = <*valor*$_2$>

- Update <tabla> set <campos>=<valores> where <condición>
  - Con esta sentencia se pretende actualizar dos campos que pertenecen a tablas diferentes siguiendo un criterio que se aplica a una tercera tabla. Al igual que para el caso anterior, no se ha obtenido un formato unificado para todos los sistemas gestores.
    - MS Access:
      - Admite la cláusula INNER JOIN de modo que el formato de la consulta resultante es el siguiente:
        - UPDATE <*tabla*$_1$> INNER JOIN (<*tabla*$_2$> INNER JOIN <*tabla*$_3$> ON <*tabla*$_2$>.<*campo*$_1$> = <*tabla*$_3$>.<*campo*$_1$>) ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_3$>.<*campo*$_x$> SET <*tabla*$_2$>.<*campo*$_v$> = <*valor*$_2$>, <*tabla*$_1$>.<*campo*$_g$> = <*valor*$_1$> WHERE <*tabla*$_3$>.<*campo*$_u$> > <*valor*$_3$>
    - Oracle:
      - Al tener que actualizar campos en las dos tablas y no admitir la cláusula INNER JOIN se ha tenido que recurrir al empleo de dos sentencias UPDATE. En ambas consultas se ha recurrido al uso de subconsultas utilizando el operador lógico ANY.
        - UPDATE <*tabla*$_1$> SET <*tabla*$_1$>.<*campo*$_g$> = <*valor*$_1$> WHERE <*tabla*$_1$>.<*campo*$_1$> = ANY

(SELECT <*campo₂*> FROM <*tabla₃*> WHERE <*tabla₃*>.<*campoᵤ*> > <*valor₃*>)

- UPDATE <*tabla₂*> SET <*tabla₂*>.<*campoᵥ*> = <*valor₂*> WHERE <*tabla2*>.<*campo1*> = ANY (SELECT <*campo₁*> FROM <*tabla₃*> WHERE <*tabla₃*>.<*campoᵤ*> > <*valor₃*>)

- MySQL:
  - Semejate a la utilizada en MS Access.
    - UPDATE <*tabla₁*> INNER JOIN (<*tabla₂*> INNER JOIN <*tabla₃*> ON <*tabla₂*>.<*campo₁*> = <*tabla₃*>.<*campo₁*>) ON <*tabla₁*>.<*campo₁*> = <*tabla₃*>.<*campoₓ*> SET <*tabla₂*>.<*campoᵥ*> = <*valor₂*>, <*tabla₁*>.<*campo_g*> = <*valor₁*> WHERE <*tabla₃*>.<*campoᵤ*> > <*valor₃*>
- Postgres:
  - No ha admitido la cláusula INNER JOIN de modo que al igual que ha ocurrido para Oracle se ha tenido que recurrir a dos consultas UPDATE.
    - UPDATE <*tabla₁*> SET <*tabla₁*>.<*campo_g*> = <*valor₁*> FROM <*tabla₃*> WHERE <tabla1>.<campo1> = <*tabla₃*>.<*campo₂*> AND <*tabla₃*>.<*campoᵤ*> > <*valor₃*>
    - UPDATE <*tabla2₁*> SET <*tabla₂*>.<*campoᵥ*> = <*valor₂*> FROM <*tabla₃*> WHERE <*tabla₂*>.<*campo₁*> = <*tabla₃*>.<*campo₁*> AND <*tabla₃*>.<*campoᵤ*> > <*valor₃*>

La tabla 1.2.7 muestra las consultas Update realizadas sobre los distintos sistemas gestores de bases de datos.

MS Access	Oracle	MySQL	Postgres
UPDATE Director SET Nombre = 'Pedro'	UPDATE Director SET Nombre = 'Pedro'	UPDATE Director SET Nombre = 'Pedro'	UPDATE Director SET Nombre = 'Pedro'
UPDATE Empleado INNER JOIN (Director INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado = DirEmpleado.id_Empleado SET Empleado.Apellidos = 'Martinez' WHERE Director.Empresa='Telefonica'	UPDATE Empleado SET Apellidos='Martinez' WHERE id_empleado=ANY (SELECT id_Empleado FROM DirEmpleado WHERE id_Director= ANY (SELECT id_Director FROM Director WHERE Empresa='Telefonica'))	UPDATE Empleado INNER JOIN (Director INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado = DirEmpleado.id_Empleado SET Empleado.Apellidos = 'Martinez' WHERE Director.Empresa='Telefonica'	UPDATE Empleado SET Apellidos='Martinez' FROM DirEmpleado WHERE DirEmpleado.id_Director= Director.id_Director AND DirEmpleado.id_Empleado= Empleado.id_Empleado AND Director.Empresa='Telefonica'
UPDATE Empleado INNER JOIN (Director	UPDATE Director SET Nombre = 'Pedro'	UPDATE Empleado INNER JOIN (Director	UPDATE Director SET Nombre = 'Pedro' FROM

INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado = DirEmpleado.id_Empleado SET Director.Nombre = 'Pedro', Empleado.Apellidos = 'Gonzalez' WHERE (((DirEmpleado.Fecha_Contrato)>#1/1/2005#))	WHERE id_Director = ANY (SELECT id_Director FROM DirEmpleado WHERE Fecha_Contrato>#1/1/2005#) UPDATE Empleado SET Apellidos = 'Gonzalez' WHERE id_Empleado = ANY (SELECT id_Empleado FROM DirEmpleado WHERE Fecha_Contrato>#1/1/2005#)	INNER JOIN DirEmpleado ON Director.id_Director = DirEmpleado.id_Director) ON Empleado.id_empleado = DirEmpleado.id_Empleado SET Director.Nombre = 'Pedro', Empleado.Apellidos = 'Gonzalez' WHERE (((DirEmpleado.Fecha_Contrato)>#1/1/2005#))	DirEmpleado WHERE DirEmpleado.id_Director = Director.id_Director AND DirEmpleado.Fecha_Contrato>#1/1/2005#) UPDATE Empleado SET Apellidos='Gonzalez' FROM DirEmpleado WHERE DirEmpleado.id_Empleado=Empleado.id_Empleado AND DirEmpleado.Fecha_Contrato>#1/1/2005#

Tabla 1.2.7 Consultas Update asociación tabla

DELETE

- Delete * from <tabla>
  - Se desea eliminar el contenido de todas las tablas. Para ello se hace uso de la integridad referencial y del borrado en cascada.
    - DELETE FROM <$tabla_2$>
- Delete where <condición>
  - En este caso se desea eliminar todos aquellos registros de una de las tablas que cumplan un cierto criterio, y mediante un borrado en cascada para mantener la integridad referencial, se eliminan los registros de la tabla que aloja la clave principal con las que está relacionado. La consulta es igual para todos los gestores.
    - DELETE FROM <$tabla_1$> WHERE <$tabla_1$>.<$campo_x$> = <$valor_1$>

La tabla 1.2.8 muestra las consultas Delete realizadas sobre los distintos sistemas gestores de bases de datos

MS Access	Oracle	MySQL	Postgres
DELETE FROM Director	DELETE FROM Director	DELETE FROM Director	DELETE FROM Director
DELETE FROM Director WHERE (((Director.Empresa) ='Telefonica'))	DELETE FROM Director WHERE (((Director.Empresa) ='Telefonica'))	DELETE FROM Director WHERE (((Director.Empresa) ='Telefonica'))	DELETE FROM Director WHERE (((Director.Empresa) ='Telefonica'))

Tabla 1.2.8 Consultas Delete asociación tabla

1.3 Herencia

La figura 1.3.1 muestra el diagrama de clases de la herencia.

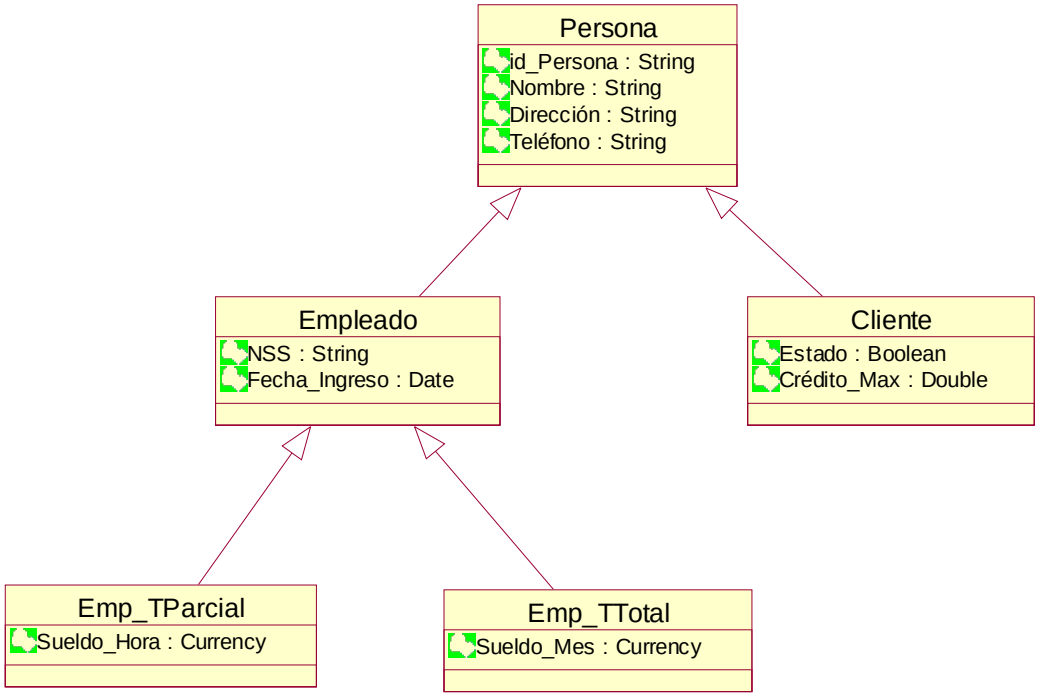


Figura 1.3.1

La herencia presenta tres formas de mapeo: mapeo por una herencia de árbol - una tabla, una clase - una tabla y una ruta de herencia – una tabla. Se muestran dichos mapeos y sus consultas asociadas.

- *Mapeo por una herencia de árbol – una tabla.* La figura 1.3.2 muestra la tabla relacional consecuencia del mapeo en *una herencia de árbol – una tabla* de la herencia.

- Una herencia de árbol - una tabla

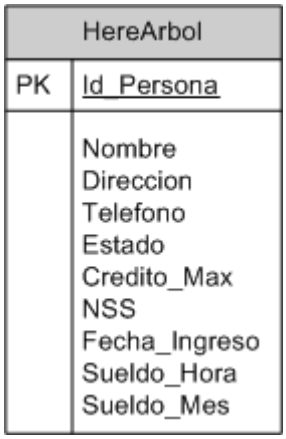


Figura 1.3.2

SELECT

- Select * from <tabla>
  - Se van a mostrar todos los campos y todos los registros de la tabla. La sentencia es compatible con todos los sistemas gestores.
    - SELECT <*tabla*>.* FROM <*tabla*>
- Select <campos> from <tabla>
  - Al igual que el caso anterior, se van a mostrar todos los registros, pero ahora de unos campos en concreto. Todos los sistemas gestores han admitido la misma consulta.
    - SELECT <*tabla*>.<$campo_x$>, …, <*tabla*>.<$campo_h$> FROM <*tabla*>
- Select * from <tabla> where <condición>
  - Se desea mostrar todos los campos de los registros que cumplen un criterio. El formato ha sido admitido por todos los sistemas gestores.
    - SELECT * FROM <*tabla*> WHERE <$campo_x$> < <*valor*>
- Select <campos> from <tabla> where <condición>
  - Se muestran determinados campos de los registros que cumplan cierto criterio. Todos los sistemas gestores han admitido la misma consulta.
    - SELECT <*tabla*>.<$campo_x$>, …, <*tabla*>.<$campo_h$> FROM <*tabla*> WHERE <$campo_x$> < <*valor*>

La tabla 1.3.1 muestra las consultas Select realizadas sobre los distintos sistemas gestores de bases de datos

MS Access	Oracle	MySQL	Postgres
SELECT HereArbol.*	SELECT HereArbol.*	SELECT HereArbol.*	SELECT HereArbol.*

FROM HereArbol	FROM HereArbol	FROM HereArbol	FROM HereArbol
SELECT HereArbol.Nombre, HereArbol.Fecha_Ingreso, HereArbol.Sueldo_Hora FROM HereArbol	SELECT HereArbol.Nombre, HereArbol.Fecha_Ingreso, HereArbol.Sueldo_Hora FROM HereArbol	SELECT HereArbol.Nombre, HereArbol.Fecha_Ingreso, HereArbol.Sueldo_Hora FROM HereArbol	SELECT HereArbol.Nombre, HereArbol.Fecha_Ingreso, HereArbol.Sueldo_Hora FROM HereArbol
SELECT HereArbol.* FROM HereArbol WHERE (((HereArbol.Sueldo_Hora)>1000))	SELECT HereArbol.* FROM HereArbol WHERE (((HereArbol.Sueldo_Hora)>1000))	SELECT HereArbol.* FROM HereArbol WHERE (((HereArbol.Sueldo_Hora)>1000))	SELECT HereArbol.* FROM HereArbol WHERE (((HereArbol.Sueldo_Hora)>1000))
SELECT HereArbol.Nombre, HereArbol.Fecha_Ingreso, HereArbol.Sueldo_Hora FROM HereArbol WHERE (((HereArbol.Sueldo_Hora)>1000))	SELECT HereArbol.Nombre, HereArbol.Fecha_Ingreso, HereArbol.Sueldo_Hora FROM HereArbol WHERE (((HereArbol.Sueldo_Hora)>1000))	SELECT HereArbol.Nombre, HereArbol.Fecha_Ingreso, HereArbol.Sueldo_Hora FROM HereArbol WHERE (((HereArbol.Sueldo_Hora)>1000))	SELECT HereArbol.Nombre, HereArbol.Fecha_Ingreso, HereArbol.Sueldo_Hora FROM HereArbol WHERE (((HereArbol.Sueldo_Hora)>1000))

Tabla 1.3.1 Consultas Select árbol de herencia

INSERT

- Insert into <tabla> values <valores>
  - Se va a insertar un registro completo en la tabla. La sentencia es admitida por todos los sistemas gestores.

    - INSERT INTO <*tabla*> VALUES (<$valor_1$>, <$valor_2$>, …, <$valor_n$>)

La tabla 1.3.2 muestra las consultas Insert realizadas sobre los distintos sistemas gestores de bases de datos

MS Access	Oracle	MySQL	Postgres
INSERT INTO HereArbol VALUES (123456789, 'Pedro','Ctra Irun 5','555161718',0,50000, 'que es esto',#25/11/2000#, 20, 3200)	INSERT INTO HereArbol VALUES (123456789, 'Pedro','Ctra Irun 5','555161718',0,50000, 'que es esto',#25/11/2000#, 20, 3200)	INSERT INTO HereArbol VALUES (123456789, 'Pedro','Ctra Irun 5','555161718',0,50000, 'que es esto',#25/11/2000#, 20, 3200)	INSERT INTO HereArbol VALUES (123456789, 'Pedro','Ctra Irun 5','555161718','0',50000, 'que es esto',#25/11/2000#, 20, 3200)

Tabla 1.3.2 Consultas Insert árbol de herencia

UPDATE

- Update <tabla> set <campo> = <valor>
  - Se actualiza un determinado campo sin aplicar ningún criterio. La consulta ha sido admitida por todos los sistemas gestores.

- UPDATE <*tabla*> SET <*campox*> = <*valor*>

o Update <tabla> set <campo> = <valor> where <condición>
  - Se actualiza un campo de todos los registros que cumplan un determinado criterio. La consulta ha sido admitida por todos los sistemas gestores.

    - UPDATE <*tabla*> SET <$campo_x$> = <$valor_1$> WHERE <$campo_2$> = <$valor_2$>

o Update <tabla> set <campos> = <valores> where <condición>
  - Se actualizan varios campos de los registros que cumplan una determinada condición. La consulta es compatible con todos los sistemas gestores.

    - UPDATE <*tabla*> SET <*tabla*>.<$campos_v$> = <$valor_1$>, <*tabla*>.<$campo_u$> = <$valor_2$> WHERE <*tabla*>.<$campo_g$> = <$valor_1$>

La tabla 1.3.3 muestra las consultas Update realizadas sobre los distintos sistemas gestores de bases de datos

MS Access	Oracle	MySQL	Postgres
UPDATE HereArbol SET Nombre = 'Fuera Zona'	UPDATE HereArbol SET Nombre = 'Fuera Zona'	UPDATE HereArbol SET Nombre = 'Fuera Zona'	UPDATE HereArbol SET Nombre = 'Fuera Zona'
UPDATE HereArbol SET Nombre = 'Fuera Zona' WHERE Sueldo_Mes=123456789	UPDATE HereArbol SET Nombre = 'Fuera Zona' WHERE Sueldo_Mes=123456789	UPDATE HereArbol SET Nombre = 'Fuera Zona' WHERE Sueldo_Mes=123456789	UPDATE HereArbol SET Nombre = 'Fuera Zona' WHERE Sueldo_Mes=123456789
UPDATE HereArbol SET Nombre = 'Fuera Zona', NSS = 'Que es esto' WHERE (((Sueldo_Mes)=123456789))	UPDATE HereArbol SET Nombre = 'Fuera Zona', NSS = 'Que es esto' WHERE (((Sueldo_Mes)=123456789))	UPDATE HereArbol SET Nombre = 'Fuera Zona', NSS = 'Que es esto' WHERE (((Sueldo_Mes)=123456789))	UPDATE HereArbol SET Nombre = 'Fuera Zona', NSS = 'Que es esto' WHERE (((Sueldo_Mes)=123456789))

Tabla 1.3.3 Consultas Update árbol de herencia

DELETE:

o Delete * from <tabla>
  - Se elimina el contenido de la tabla.

    - DELETE FROM <*tabla*>

o Delete * from <tabla> where <condición>
  - Borrado similar al anterior, pero con un criterio.

    - DELETE FROM <*tabla*> WHERE <$campo_x$> = <*valor*>

La tabla 1.3.4 muestra las consultas Delete realizadas sobre los distintos sistemas gestores de bases de datos

MS Access	Oracle	MySQL	Postgres
DELETE FROM HereArbol	DELETE FROM HereArbol	DELETE FROM HereArbol	DELETE FROM HereArbol
DELETE FROM HereArbol WHERE (((HereArbol.Sueldo_Mes)=123456789))	DELETE FROM HereArbol WHERE (((HereArbol.Sueldo_Mes)=123456789))	DELETE FROM HereArbol WHERE (((HereArbol.Sueldo_Mes)=123456789))	DELETE FROM HereArbol WHERE (((HereArbol.Sueldo_Mes)=123456789))

Tabla 1.3.4 Consultas Delete un árbol de herencia – una tabla

- Una clase - una tabla

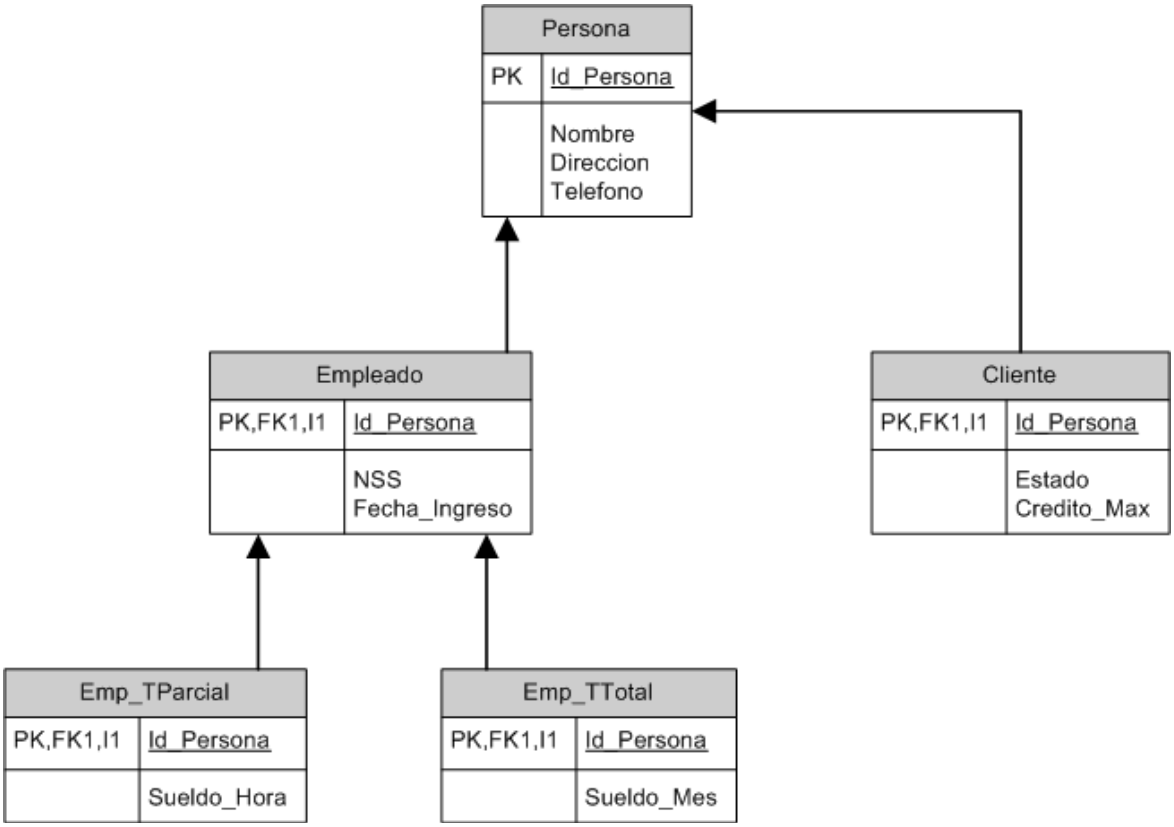


Figura 1.3.3

SELECT

- Select * from <tablas>
  - Se desean obtener todos los registros. Al tratarse de una herencia, se emplea la cláusula LEFT JOIN para vincular las. Esta consulta es compatible con todos los sistemas gestores salvo MS Access, en cuyo caso presenta una sintaxis diferente.

- MS Access:
  - SELECT <*tabla*$_1$>.*, … , <*tabla*$_n$>.* FROM (<*tabla*$_1$> LEFT JOIN <*tabla*$_2$> ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_2$>.<*campo*$_1$>) LEFT JOIN ((<*tabla*$_3$> LEFT JOIN <*tabla*$_4$> ON <*tabla*$_3$>.<*campo*$_1$> = <*tabla*$_4$>.<*campo*$_1$>) LEFT JOIN <*tabla*$_5$> ON <*tabla*$_3$>.<*campo*$_1$> = <*tabla*$_5$>.<*campo*$_1$>) ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_3$>.<*campo*$_1$>
- Resto:
  - SELECT <*tabla*$_1$>.*, …, <*tabla*$_n$>. * FROM <*tabla*$_1$> LEFT JOIN <*tabla*$_2$> ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_2$>.<*campo*$_1$> LEFT JOIN <*tabla*$_3$>.<*campo*$_1$> ON <*tabla*$_1$>.<*campo*$_1$> …

- Select <campos> from <tablas>
  - Es una consulta de proyección que vuelve a utilizar la cláusula LEFT JOIN. Todos los sistemas gestores han admitido el mismo formato salvo Access.
    - MS Access:
      - SELECT <*tabla*$_1$>.<*campo*$_x$>, … , <*tabla*$_d$>.<*campo*$_h$> FROM (<*tabla*$_1$> LEFT JOIN <*tabla*$_2$> ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_2$>.<*campo*$_1$>) LEFT JOIN ((<*tabla*$_3$> LEFT JOIN <*tabla*$_4$> ON <*tabla*$_3$>.<*campo*$_1$> = <*tabla*$_4$>.<*campo*$_1$>) LEFT JOIN <*tabla*$_5$> ON <*tabla*$_3$>.<*campo*$_1$> = <*tabla*$_5$>.<*campo*$_1$>) ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_3$>.<*campo*$_1$>
    - Resto:
      - SELECT <*tabla*$_1$>.<*campo*$_x$>, … , <*tabla*$_d$>.<*campo*$_h$> FROM <*tabla*$_1$> LEFT JOIN <*tabla*$_2$> ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_2$>.<*campo*$_1$> LEFT JOIN <*tabla*$_3$>.<*campo*$_1$> ON <*tabla*$_1$>.<*campo*$_1$> …

- Select * from <tablas> where <condición>
  - Se trata de una consulta de selección en funcion de un criterio. Se muestran todos los campos de todas las tablas cuyos registros cumplan un cierto criterio. Para ello se utiliza una consulta con un formato semejante al empleado para el 'select * from <tabla>' salvo que se le añade un 'where <condición>'. Todos los sistemas gestores han admitido el mismo formato salvo Access.
    - MS Access:
      - SELECT <*tabla*$_1$>.*, … , <*tabla*$_n$>.* FROM (<*tabla*$_1$> LEFT JOIN <*tabla*$_2$> ON <*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_2$>.<*campo*$_1$>) LEFT JOIN ((<*tabla*$_3$> LEFT JOIN <*tabla*$_4$> ON <*tabla*$_3$>.<*campo*$_1$> = <*tabla*$_4$>.<*campo*$_1$>) LEFT JOIN <*tabla*$_5$> ON <*tabla*$_3$>.<*campo*$_1$> = <*tabla*$_5$>.<*campo*$_1$>) ON

<*tabla₁*>.<*campo₁*> = <*tabla₃*>.<*campo₁*> WHERE <*tabla_g*>.<*campo_r*> > <*valor*>

- Resto:
  - SELECT <*tabla₁*>.*, …, <*tabla_n*>. * FROM <*tabla₁*> LEFT JOIN <*tabla₂*> ON <*tabla₁*>.<*campo₁*> = <*tabla₂*>.<*campo₁*> LEFT JOIN <*tabla₃*>.<*campo₁*> ON <*tabla₁*>.<*campo₁*> … WHERE <*tabla_g*>.<*campo_r*> > <*valor*>

- Select <campos> from <tablas> where <condición>
  - Consulta similar a la anterior en la que se realiza una proyección.
    - MS Access:
      - SELECT <*tabla₁*>.<*campo_x*>, … , <*tabla_d*>.<*campo_h*> FROM (<*tabla₁*> LEFT JOIN <*tabla₂*> ON <*tabla₁*>.<*campo₁*> = <*tabla₂*>.<*campo₁*>) LEFT JOIN ((<*tabla₃*> LEFT JOIN <*tabla₄*> ON <*tabla₃*>.<*campo₁*> = <*tabla₄*>.<*campo₁*>) LEFT JOIN <*tabla₅*> ON <*tabla₃*>.<*campo₁*> = <*tabla₅*>.<*campo₁*>) ON <*tabla₁*>.<*campo₁*> = <*tabla₃*>.<*campo₁*> WHERE <*tabla_g*>.<*campo_r*> > <*valor*>

    - Resto:
      - SELECT <*tabla₁*>.<*campo_x*>, … , <*tabla_d*>.<*campo_h*> FROM <*tabla₁*> LEFT JOIN <*tabla₂*> ON <*tabla₁*>.<*campo₁*> = <*tabla₂*>.<*campo₁*> LEFT JOIN <*tabla₃*>.<*campo₁*> ON <*tabla₁*>.<*campo₁*> … WHERE <*tabla_g*>.<*campo_r*> > <*valor*>

La tabla 1.3.5 muestra las consultas Select realizadas sobre los distintos sistemas gestores de bases de datos

MS Access	Oracle	MySQL	Postgres
SELECT Persona.*, Cliente.*,Empleado.*, Emp_TTotal.*, Emp_TParcial.* FROM (Persona LEFT JOIN Cliente ON Persona.Id_Persona = Cliente.Id_Persona) LEFT JOIN ((Empleado LEFT JOIN Emp_TTotal ON Empleado.Id_Persona = Emp_TTotal.Id_Persona) LEFT JOIN Emp_TParcial ON Empleado.Id_Persona = Emp_TParcial.Id_Person	SELECT Persona.*, Cliente.*,Empleado.*, Emp_TTotal.*, Emp_TParcial.* FROM Persona LEFT JOIN cliente on Persona.Id_Persona=cliente.Id_Persona LEFT JOIN empleado ON Persona.Id_Persona=empleado.Id_Persona LEFT JOIN Emp_TTotal ON empleado.Id_Persona=Emp_Ttotal.Id_Persona LEFT JOIN Emp_TParcial ON empleado.Id_Persona=E	SELECT Persona.*, Cliente.*,Empleado.*, Emp_TTotal.*, Emp_TParcial.* FROM Persona LEFT JOIN cliente on Persona.Id_Persona=cliente.Id_Persona LEFT JOIN empleado ON Persona.Id_Persona=empleado.Id_Persona LEFT JOIN Emp_TTotal ON empleado.Id_Persona=Emp_Ttotal.Id_Persona LEFT JOIN Emp_TParcial ON empleado.Id_Persona=E	SELECT Persona.*, Cliente.*,Empleado.*, Emp_TTotal.*, Emp_TParcial.* FROM Persona LEFT JOIN cliente on Persona.Id_Persona=cliente.Id_Persona LEFT JOIN empleado ON Persona.Id_Persona=empleado.Id_Persona LEFT JOIN Emp_TTotal ON empleado.Id_Persona=Emp_Ttotal.Id_Persona LEFT JOIN Emp_TParcial ON empleado.Id_Persona=E

a) ON Persona.Id_Persona = Empleado.Id_Persona	mp_TParcial.Id_Persona	mp_TParcial.Id_Persona	mp_TParcial.Id_Persona
SELECT Cliente.Id_Persona, Persona.Nombre, Cliente.Credito_Max, Emp_TParcial.Sueldo_Hora, Emp_TTotal.Sueldo_Mes FROM (((Persona LEFT JOIN Empleado ON Persona.Id_Persona = Empleado.Id_Persona) LEFT JOIN Cliente ON Persona.Id_Persona = Cliente.Id_Persona) LEFT JOIN Emp_TParcial ON Empleado.Id_Persona = Emp_TParcial.Id_Persona) LEFT JOIN Emp_TTotal ON Empleado.Id_Persona = Emp_TTotal.Id_Persona	SELECT Persona.Nombre, Empleado.Fecha_Ingreso, Emp_TParcial.Sueldo_Hora FROM Persona LEFT JOIN cliente on Persona.Id_Persona=cliente.Id_Persona LEFT JOIN empleado ON Persona.Id_Persona=empleado.Id_Persona LEFT JOIN Emp_TTotal ON empleado.Id_Persona=Emp_Ttotal.Id_Persona LEFT JOIN Emp_TParcial ON empleado.Id_Persona=Emp_TParcial.Id_Persona	SELECT Persona.Nombre, Empleado.Fecha_Ingreso, Emp_TParcial.Sueldo_Hora FROM Persona LEFT JOIN cliente on Persona.Id_Persona=cliente.Id_Persona LEFT JOIN empleado ON Persona.Id_Persona=empleado.Id_Persona LEFT JOIN Emp_TTotal ON empleado.Id_Persona=Emp_Ttotal.Id_Persona LEFT JOIN Emp_TParcial ON empleado.Id_Persona=Emp_TParcial.Id_Persona	SELECT Persona.Nombre, Empleado.Fecha_Ingreso, Emp_TParcial.Sueldo_Hora FROM Persona LEFT JOIN cliente on Persona.Id_Persona=cliente.Id_Persona LEFT JOIN empleado ON Persona.Id_Persona=empleado.Id_Persona LEFT JOIN Emp_TTotal ON empleado.Id_Persona=Emp_Ttotal.Id_Persona LEFT JOIN Emp_TParcial ON empleado.Id_Persona=Emp_TParcial.Id_Persona
SELECT Persona.*, Empleado.*, Cliente.*, Emp_TParcial.*, Emp_TTotal.* FROM (Persona LEFT JOIN Cliente ON Persona.Id_Persona = Cliente.Id_Persona) LEFT JOIN ((Empleado LEFT JOIN Emp_TTotal ON Empleado.Id_Persona = Emp_TTotal.Id_Persona) LEFT JOIN Emp_TParcial ON Empleado.Id_Persona = Emp_TParcial.Id_Persona) ON Persona.Id_Persona = Empleado.Id_Persona WHERE (((Emp_TParcial.Sueldo_Hora)>1000))	SELECT Persona.*, Empleado.*, Cliente.*, Emp_TParcial.*, Emp_TTotal.* FROM (Persona LEFT JOIN Cliente ON Persona.Id_Persona = Cliente.Id_Persona) LEFT JOIN ((Empleado LEFT JOIN Emp_TTotal ON Empleado.Id_Persona = Emp_TTotal.Id_Persona) LEFT JOIN Emp_TParcial ON Empleado.Id_Persona = Emp_TParcial.Id_Persona) ON Persona.Id_Persona = Empleado.Id_Persona WHERE (((Emp_TParcial.Sueldo_Hora)>1000))	SELECT Persona.*, Empleado.*, Cliente.*, Emp_TParcial.*, Emp_TTotal.* FROM (Persona LEFT JOIN Cliente ON Persona.Id_Persona = Cliente.Id_Persona) LEFT JOIN ((Empleado LEFT JOIN Emp_TTotal ON Empleado.Id_Persona = Emp_TTotal.Id_Persona) LEFT JOIN Emp_TParcial ON Empleado.Id_Persona = Emp_TParcial.Id_Persona) ON Persona.Id_Persona = Empleado.Id_Persona WHERE (((Emp_TParcial.Sueldo_Hora)>1000))	SELECT Persona.*, Empleado.*, Cliente.*, Emp_TParcial.*, Emp_TTotal.* FROM (Persona LEFT JOIN Cliente ON Persona.Id_Persona = Cliente.Id_Persona) LEFT JOIN ((Empleado LEFT JOIN Emp_TTotal ON Empleado.Id_Persona = Emp_TTotal.Id_Persona) LEFT JOIN Emp_TParcial ON Empleado.Id_Persona = Emp_TParcial.Id_Persona) ON Persona.Id_Persona = Empleado.Id_Persona WHERE (((Emp_TParcial.Sueldo_Hora)>1000))
SELECT Persona.Nombre, Empleado.Fecha_Ingreso, Emp_TParcial.Sueldo_Hora FROM (Persona LEFT JOIN Cliente ON Persona.Id_Persona = Cliente.Id_Persona) LEFT JOIN ((Empleado	SELECT Persona.Nombre, Empleado.Fecha_Ingreso, Emp_TParcial.Sueldo_Hora FROM Persona LEFT JOIN cliente on Persona.Id_Persona=cliente.Id_Persona LEFT JOIN empleado ON	SELECT Persona.Nombre, Empleado.Fecha_Ingreso, Emp_TParcial.Sueldo_Hora FROM Persona LEFT JOIN cliente on Persona.Id_Persona=cliente.Id_Persona LEFT JOIN empleado ON	SELECT Persona.Nombre, Empleado.Fecha_Ingreso, Emp_TParcial.Sueldo_Hora FROM Persona LEFT JOIN cliente on Persona.Id_Persona=cliente.Id_Persona LEFT JOIN empleado ON

LEFT JOIN Emp_TTotal ON Empleado.Id_Persona = Emp_TTotal.Id_Persona) LEFT JOIN Emp_TParcial ON Empleado.Id_Persona = Emp_TParcial.Id_Persona) ON Persona.Id_Persona = Empleado.Id_Persona WHERE (((Emp_TTotal.Sueldo_Mes)>1000))	Persona.Id_Persona=empleado.Id_Persona LEFT JOIN Emp_TTotal ON empleado.Id_Persona=Emp_Ttotal.Id_Persona LEFT JOIN Emp_TParcial ON empleado.Id_Persona=Emp_TParcial.Id_Persona WHERE emp_TParcial.sueldo_Hora>1000	Persona.Id_Persona=empleado.Id_Persona LEFT JOIN Emp_TTotal ON empleado.Id_Persona=Emp_Ttotal.Id_Persona LEFT JOIN Emp_TParcial ON empleado.Id_Persona=Emp_TParcial.Id_Persona WHERE emp_TParcial.sueldo_Hora>1000	Persona.Id_Persona=empleado.Id_Persona LEFT JOIN Emp_TTotal ON empleado.Id_Persona=Emp_Ttotal.Id_Persona LEFT JOIN Emp_TParcial ON empleado.Id_Persona=Emp_TParcial.Id_Persona WHERE emp_TParcial.sueldo_Hora>1000

Tabla 1.3.5 Consultas Select

INSERT

- Insert into <tablas> values (<valores>)
  - Como se está trabajando sobre varias tablas se utilizan tres sentencias Insert consecutivas. Todos los sistemas gestores han admitido el mismo formato.

    - INSERT INTO <$tabla_1$> VALUES (<$valores_1$>)
    - INSERT INTO <$tabla_2$> VALUES (<$valores_2$>)
    - INSERT INTO <$tabla_4$> VALUES (<$valores_3$>)

La tabla 1.3.6 muestra las consultas Insert realizadas sobre los distintos sistemas gestores de bases de datos

MS Access	Oracle	MySQL	Postgres
INSERT INTO Persona VALUES (123456789,'Pedro','Ctra Irun 5','555161718') INSERT INTO Empleado VALUES (123456789,'que es esto',#25/11/2000#) INSERT INTO Emp_TParcial VALUES (123456789,20)	INSERT INTO Persona VALUES (123456789,'Pedro','Ctra Irun 5','555161718') INSERT INTO Empleado VALUES (123456789,'que es esto',#25/11/2000#) INSERT INTO Emp_TParcial VALUES (123456789,20)	INSERT INTO Persona VALUES (123456789,'Pedro','Ctra Irun 5','555161718') INSERT INTO Empleado VALUES (123456789,'que es esto',#25/11/2000#) INSERT INTO Emp_TParcial VALUES (123456789,20)	INSERT INTO Persona VALUES (123456789,'Pedro','Ctra Irun 5','555161718') INSERT INTO Empleado VALUES (123456789,'que es esto',#25/11/2000#) INSERT INTO Emp_TParcial VALUES (123456789,20)

Tabla 1.3.6 Consultas Insert árbol de herencia

UPDATE

- Update <tabla> set <campo> = <valor>
  - Se actualiza un determinado campo sin aplicar ningún criterio. La sentencia es compatible con todos los sistemas gestores.

    - UPDATE <$tabla_1$> SET <$campo_x$> = <*valor*>

- Update <tabla> set <campo> = <valor> where <condición>
  - Se actualiza un campo que cumple un determinado criterio. El campo que se desea actualizar no se encuentra en la tabla sobre la que se aplica el criterio de actualización. Todos los sistemas gestores no admiten el mismo formato.
    - MS Access:
      - Al poder hacer uso de INNER JOIN, el formato de la consulta es simple y se puede realizar en una sola consulta.
        - UPDATE <$tabla_1$> INNER JOIN <$tabla_5$> ON <$tabla_1$>.<$campo_1$> = <$tabla_5$>.<$campo_5$> SET <$tabla_1$>.<$campo_x$> = <*valor1*> WHERE <$tabla_5$>.<$campo_h$> = <$valor_2$>
    - Oracle:
      - Como ya se ha realizado en casos anteriores, se ha recurrido al uso de una subconsulta SELECT empleando la cláusula ANY. Esta subconsulta incluye el criterio de actualización.
        - UPDATE <$tabla_1$> SET <$tabla_1$>.<$campo_x$> = <*valor1*> WHERE <$campo_1$> = ANY (SELECT <$campo_1$> FROM <$tabla_5$> WHERE <$tabla_5$>.<$campo_h$> = <$valo_2$>)
    - MySQL:
      - Admite INNER JOIN del mismo modo que lo hace Access, de modo que la consulta resultante tiene el mismo formato en ambos sistemas gestores.
        - UPDATE <$tabla_1$> INNER JOIN <$tabla_5$> ON <$tabla_1$>.<$campo_1$> = <$tabla_5$>.<$campo_5$> SET <$tabla_1$>.<$campo_x$> = <*valor1*> WHERE <$tabla_5$>.<$campo_h$> = <$valor_2$>
    - Postgres:
      - Se ha empleado una consulta UPDATE, pero con dos condiciones encadenas por el operador lógico AND. La primera condición se limita a que la clave principal de las tablas sean iguales y la segunda contiene l criterio de actualización
        - UPDATE <$tabla_1$> SET <$tabla_1$>.<$campo_x$> = <*valor1*> FROM <*tabla5*> WHERE <$tabla_1$>.<$campo_1$> = <$tabla_5$>.<$campo_1$> AND <$tabla_5$>.<$campo_x$> = <$valor_2$>

- Update <tabla> set <campos> = <valores> where <condición>
  - Se actualizan varios campos que cumplan una determinada condición. Se han actualizado dos campos de dos tablas pertenecientes a la misma rama, aplicando el criterio a una tercera tabla de esa misma rama.
    - MS Access:
      - Al poder hacer uso de INNER JOIN el formato de la consulta queda de la siguiente manera:
        - UPDATE <$tabla_1$> INNER JOIN (<$tabla_2$> INNER JOIN <$tabla_5$> ON <$tabla_2$>.<$campo_1$> = <$tabla_5$>.<$campo_5$>) ON <$tabla_1$>.<$campo_1$> = <$tabla_2$>.<$campo_1$> SET <$tabla_1$>.<$campo_x$> = <valor1> , <$tabla_2$>.<$campo_y$> = <$valor_3$> WHERE <$tabla_5$>.<$campo_h$> = <$valor_2$>
    - Oracle:
      - Se han utilizado dos consultas UPDATE consecutivas. Ambas consultas cuentan con una subconsulta enlazada con ANY que añade como criterio la igualdad entre las claves principales. Esta subconsulta implementa la condición de actualización.
        - UPDATE <$tabla_1$> SET <$campo_x$> = <$valor_1$> WHERE <$campo_1$> = ANY (SELECT <$campo_1$> FROM <$tabla_5$> WHERE<$tabla_5$>.<$campo_h$> = <$valor_2$>)
        - UPDATE <$tabla_2$> SET <$campo_y$> = <$valor_3$> WHERE <$campo_1$> = ANY (SELECT <$campo_1$> FROM <$tabla_5$> WHERE<$tabla_5$>.<$campo_h$> = <$valor_2$>)
    - MySQL:
      - Semejante a MS Access.
        - UPDATE <$tabla_1$> INNER JOIN (<$tabla_2$> INNER JOIN <$tabla_5$> ON <$tabla_2$>.<$campo_1$> = <$tabla_5$>.<$campo_5$>) ON <$tabla_1$>.<$campo_1$> = <$tabla_2$>.<$campo_1$> SET <$tabla_1$>.<$campo_x$> = <valor1> , <$tabla_2$>.<$campo_y$> = <$valor_3$> WHERE <$tabla_5$>.<$campo_h$> = <$valor_2$>
    - Postgres:
      - Se han encadenado dos consultas UPDATE cada una de las cuales incluye dos condiciones. La primera es la igualdad entre las claves principales y la segunda contiene la condición de actualización.
        - UPDATE <$tabla_1$> SET <$tabla_1$>.<$campo_x$> = <$valor_1$> FROM <$tabla_5$> WHERE

<*tabla*$_1$>.<*campo*$_1$> = <*tabla*$_5$>.<*campo*$_1$> AND <*tabla*$_5$>.<*campo*$_h$> = <*valor*$_2$>

- UPDATE <*tabla*$_2$> SET <*tabla*$_2$>.<*campo*$_y$> = <*valor*$_3$> FROM <*tabla*$_5$> WHERE <*tabla*$_2$>.<*campo*$_1$> = <*tabla*$_5$>.<*campo*$_1$> AND <*tabla*$_5$>.<*campo*$_h$> = <*valor*$_2$>

La tabla 1.3.7 muestra las consultas Update realizadas sobre los distintos sistemas gestores de bases de datos

MS Access	Oracle	MySQL	Postgres
UPDATE Persona SET Nombre = 'Fuera Zona'	UPDATE Persona SET Nombre = 'Fuera Zona'	UPDATE Persona SET Nombre = 'Fuera Zona'	UPDATE Persona SET Nombre = 'Fuera Zona'
UPDATE Persona INNER JOIN Emp_TTotal ON Persona.Id_Persona = Emp_TTotal.Id_Persona SET Persona.Nombre = 'Fuera Zona' WHERE Emp_TTotal.Sueldo_Mes=123456789	UPDATE Persona SET Nombre='Fuera Zona' WHERE id_persona= ANY (SELECT id_persona FROM emp_ttotal WHERE sueldo_mes=123456789)	UPDATE Persona INNER JOIN Emp_TTotal ON Persona.Id_Persona = Emp_TTotal.Id_Persona SET Persona.Nombre = 'Fuera Zona' WHERE Emp_TTotal.Sueldo_Mes=123456789	UPDATE Persona SET Nombre='Fuera Zona' FROM Emp_TTotal WHERE Persona.id_Persona=Emp_TTotal.id_persona AND Emp_ttotal.sueldo_mes= 123456789
UPDATE Persona INNER JOIN (Empleado INNER JOIN Emp_TTotal ON Empleado.Id_Persona = Emp_TTotal.Id_Persona) ON Persona.Id_Persona = Empleado.Id_Persona SET Persona.Nombre = 'Fuera Zona', Empleado.NSS = 'que es esto' WHERE (((Emp_TTotal.Sueldo_Mes)=123456789))	UPDATE Persona SET Nombre='Fuera Zona' WHERE id_Persona=ANY (SELECT id_Persona FROM EMP_TTotal WHERE Sueldo_Mes=123456789) UPDATE Empleado SET NSS = 'Que es esto' WHERE id_Persona=ANY (SELECT id_Persona FROM EMP_TTotal WHERE Sueldo_Mes=123456789)	UPDATE Persona INNER JOIN (Empleado INNER JOIN Emp_TTotal ON Empleado.Id_Persona = Emp_TTotal.Id_Persona) ON Persona.Id_Persona = Empleado.Id_Persona SET Persona.Nombre = 'Fuera Zona', Empleado.NSS = 'que es esto' WHERE (((Emp_TTotal.Sueldo_Mes)=123456789))	UPDATE Persona SET Nombre = 'Fuera Zona' FROM Emp_TTotal WHERE Persona.id_Persona=Emp_TTotal.id_Persona AND Emp_TTotal.Sueldo_Mes=123456789 UPDATE Empleado SET NSS = 'que es esto' FROM Emp_TTotal WHERE Empleado.id_Persona=Emp_TTotal.id_Persona AND Emp_TTotal.Sueldo_Mes=123456789

Tabla 1.3.7 Consultas Update

DELETE

- Delete * from <tabla>
  - Se eliminan todos los registros de las tablas. Para ello se comienza eliminando la tabla inicial y mediante el borrado en cascada y la integridad referencial se eliminan los registros en el resto. Es admitida por todos los sistemas gestores.

- DELETE FROM <$tabla_1$>

o Delete * from <tabla> where <condición>

- Se eliman los registros de varias tablas que pertenecen a la misma rama según la condición. El criterio de eliminanción se aplica sobre una de las tablas y se procede a eliminar los registros en el resto de tablas mediante el borrado en cascada. Para realizar esto se ha generado una consulta delete que hace uso de una subconsulta enlazada con ANY con el objeto de que sea compatbile con todos los sistemas gestores.

  - DELETE FROM <$tabla_1$> WHERE <$campo_1$> = ANY (SELECT <$campo_1$> FROM <$tabla_5$> WHERE <$campo_x$> = <*valor*>)

La tabla 1.3.8 muestra las consultas Delete realizadas sobre los distintos sistemas gestores de bases de datos

MS Access	Oracle	MySQL	Postgres
DELETE FROM Persona	DELETE FROM Persona	DELETE FROM Persona	DELETE FROM Persona
DELETE FROM Persona WHERE id_Persona=ANY (SELECT id_Persona FROM Emp_TTotal WHERE sueldo_mes=123456789)	DELETE FROM Persona WHERE id_Persona=ANY (SELECT id_Persona FROM Emp_TTotal WHERE sueldo_mes=123456789)	DELETE FROM Persona WHERE id_Persona=ANY (SELECT id_Persona FROM Emp_TTotal WHERE sueldo_mes=123456789)	DELETE FROM Persona WHERE id_Persona=ANY (SELECT id_Persona FROM Emp_TTotal WHERE sueldo_mes=123456789)

Tabla 1.3.8 Consultas Delete

- Una ruta de herencia - una tabla

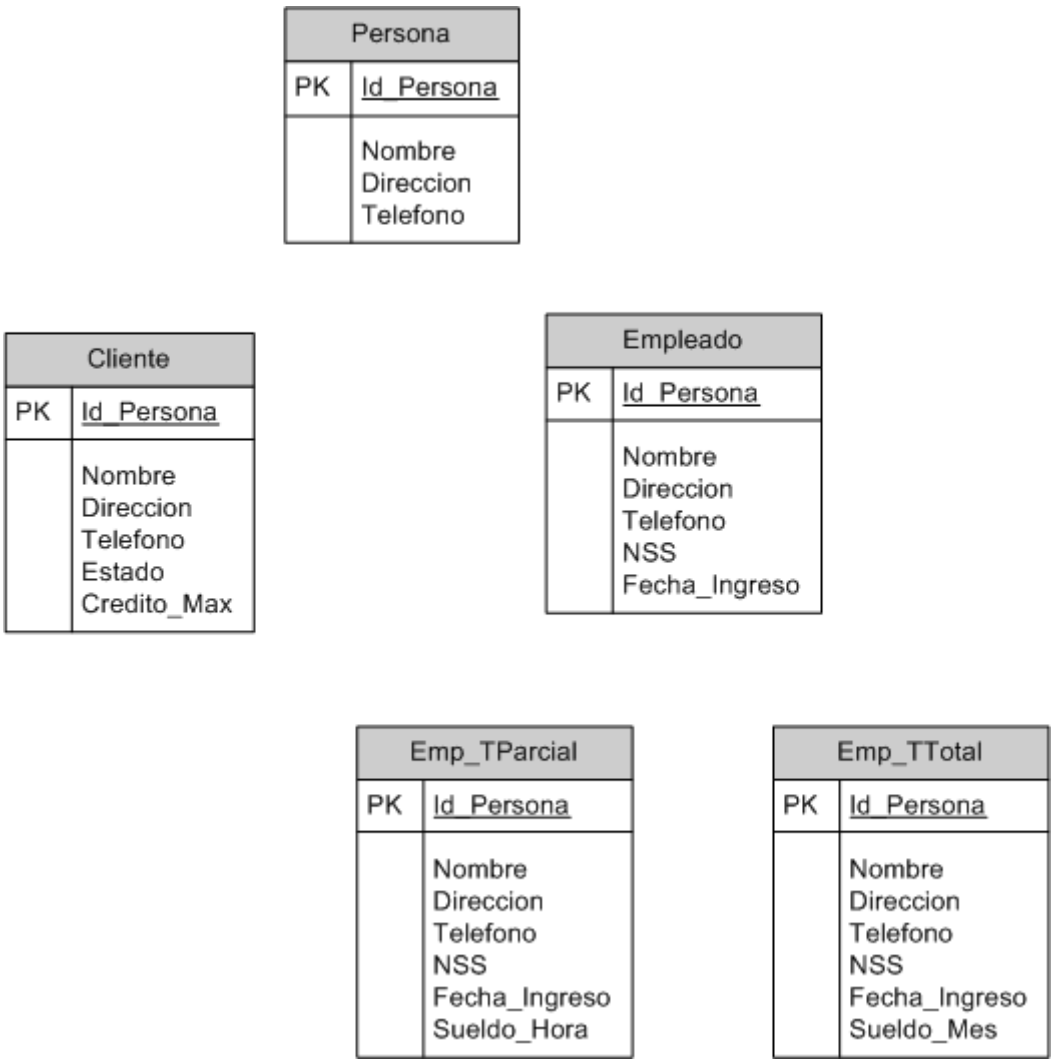


Figura 1.3.4

SELECT

- Select * from <tablas>
  - Para mostrar el contenido de todas las tablas en este mapeo se recurre al uso de varios 'select from <tablas>' concatenados. La consulta es admitida por todos los sistemas gestores.

    SELECT FROM <*tabla*$_1$>
    SELECT FROM <*tabla*$_2$>
    SELECT FROM <*tabla*$_3$>
    SELECT FROM <*tabla*$_4$>
    SELECT FROM <*tabla*$_5$>

- Select <campos> from <tablas>

- Se muestran varios campos de una de las tablas, la consulta presenta un formato compatible con todos los sistemas gestores de bases de datos y es el siguiente:
  - SELECT <*tabla*>.<*campo*$_1$>, <*tabla*>.<*campo*$_1$>, <*tabla*>.<*campo*$_1$> FROM <*tabla*>

- Select * from <tabla> where <condición>
  - Se muestran todos los campos de los registros que cumplan cierto criterio, el formato de la consulta es común para todos los sistemas gestores y es el siguiente:
    - SELECT <*tabla*>.* FROM <*tabla*> WHERE <*tabla*>.<*campo*> > <*valor*>

- Select <campos> from <tablas> where <condición>
  - Consulta similar a la anterior, pero en la que en lugar de mostrarse todos los campos solo se muestran unos en concreto. Al igual que en el caso anterior la consulta presenta un formato común para todos los sistemas gestores.
    - SELECT <*tabla*>.<*campo*$_1$>, <*tabla*>.<*campo*$_2$>, <*tabla*>.<*campo*$_3$>, FROM <*tabla*> WHERE <*tabla*>.<*campo*$_x$> > <*valor*>

La tabla 1.3.9 muestra las consultas Select realizadas sobre los distintos sistemas gestores de bases de datos.

MS Access	Oracle	MySQL	Postgres
SELECT * FROM Persona	SELECT * FROM Persona	SELECT * FROM Persona	SELECT * FROM Persona
SELECT * FROM Cliente	SELECT * FROM Cliente	SELECT * FROM Cliente	SELECT * FROM Cliente
SELECT * FROM Empleado	SELECT * FROM Empleado	SELECT * FROM Empleado	SELECT * FROM Empleado
SELECT * FROM Emp_TTotal	SELECT * FROM Emp_TTotal	SELECT * FROM Emp_TTotal	SELECT * FROM Emp_TTotal
SELECT * FROM Emp_TParcial	SELECT * FROM Emp_TParcial	SELECT * FROM Emp_TParcial	SELECT * FROM Emp_TParcial
SELECT Emp_TParcial.Nombre, Emp_TParcial.Fecha_Ingreso, Emp_TParcial.Sueldo_Hora FROM Emp_TParcial	SELECT Emp_TParcial.Nombre, Emp_TParcial.Fecha_Ingreso, Emp_TParcial.Sueldo_Hora FROM Emp_TParcial	SELECT Emp_TParcial.Nombre, Emp_TParcial.Fecha_Ingreso, Emp_TParcial.Sueldo_Hora FROM Emp_TParcial	SELECT Emp_TParcial.Nombre, Emp_TParcial.Fecha_Ingreso, Emp_TParcial.Sueldo_Hora FROM Emp_TParcial
SELECT Emp_TParcial.* FROM Emp_TParcial WHERE Emp_TParcial.Sueldo_Hora>1000	SELECT Emp_TParcial.* FROM Emp_TParcial WHERE Emp_TParcial.Sueldo_Hora>1000	SELECT Emp_TParcial.* FROM Emp_TParcial WHERE Emp_TParcial.Sueldo_Hora>1000	SELECT Emp_TParcial.* FROM Emp_TParcial WHERE Emp_TParcial.Sueldo_Hora>1000

SELECT Emp_TParcial.Nombre, Emp_TParcial.Fecha_Ingreso, Emp_TParcial.Sueldo_Hora FROM Emp_TParcial WHERE Emp_TParcial.Sueldo_Hora>1000	SELECT Emp_TParcial.Nombre, Emp_TParcial.Fecha_Ingreso, Emp_TParcial.Sueldo_Hora FROM Emp_TParcial WHERE Emp_TParcial.Sueldo_Hora>1000	SELECT Emp_TParcial.Nombre, Emp_TParcial.Fecha_Ingreso, Emp_TParcial.Sueldo_Hora FROM Emp_TParcial WHERE Emp_TParcial.Sueldo_Hora>1000	SELECT Emp_TParcial.Nombre, Emp_TParcial.Fecha_Ingreso, Emp_TParcial.Sueldo_Hora FROM Emp_TParcial WHERE Emp_TParcial.Sueldo_Hora>1000

Tabla 1.3.9 Consultas Select

INSERT

- Insert into <tablas> values (<valores>)
  - Se procede a la inserción de un nuevo objeto en la base de datos, dado que los objetos están separados en tablas independientes tan solo se deberá realizar una consulta de inserción sobre una de las tablas.

    - INSERT INTO <$tabla_1$> VALUES (<$valores_1$>)

La tabla X.25 muestra las consultas Insert realizadas sobre los distintos sistemas gestores de bases de datos

MS Access	Oracle	MySQL	Postgres
INSERT INTO Emp_TParcial VALUES (123456789,'Pedro', 'Ctra Irun 5', '555161718', 'que es esto',#25/11/2000#,20)	INSERT INTO Emp_TParcial VALUES (123456789,'Pedro', 'Ctra Irun 5', '555161718', 'que es esto',#25/11/2000#,20)	INSERT INTO Emp_TParcial VALUES (123456789,'Pedro', 'Ctra Irun 5', '555161718', 'que es esto',#25/11/2000#,20)	INSERT INTO Emp_TParcial VALUES (123456789,'Pedro', 'Ctra Irun 5', '555161718', 'que es esto',#25/11/2000#,20)

Tabla 1.3.9 Consultas Insert

UPDATE

- Update <tabla> set <campo> = <valor>
  - Se actualiza un determinado en una tabla. Al tratarse de un mapeo de herencia una ruta de herencia - una tabla, este campo se ha de actualizar en todas las tablas que se situan en la misma ruta de herencia. Para ello se utilizan una serie de consultas UPDATE que actualizan el mismo campo en las tablas implicadas.Todos los sistemas gestores admiten el mismo formato.

    - UPDATE <$tabla_1$> SET <$campo_x$> = <*valor*>
    - UPDATE <$tabla_2$> SET <$campo_y$> = <*valor*>
    - UPDATE <$tabla_3$> SET <$campo_v$> = <*valor*>
    - UPDATE <$tabla_4$> SET <$campo_u$> = <*valor*>
    - UPDATE <$tabla_5$> SET <$campo_z$> = <*valor*>

- Update <tabla> set <campo> = <valor> where <condición>

- Se actualiza un campo en función de un criterio. Dado que los objetos a actualizar se encuentran en una tabla, la consulta resultante es la siguiente:
  - UPDATE <*tabla*> SET <$campo_1$>=<$valor_1$> WHERE <$campo_2$>=<$valor_2$>

- Update <tabla> set <campos> = <valores> where <condición>
  - Similar a la consulta anterior, pero en este caso se actualizan varios campos.
    - UPDATE <*tabla*> SET <$campo_1$>=<$valor_1$>, <$campo_2$>=<$valor_2$> WHERE <$campo_3$>=<$valor_3$>

La tabla 1.3.10 muestra las consultas Update realizadas sobre los distintos sistemas gestores de bases de datos

MS Access	Oracle	MySQL	Postgres
UPDATE Persona SET Nombre='Fuera Zona' UPDATE Empleado SET Nombre='Fuera Zona' UPDATE Emp_TParcial SET Nombre='Fuera Zona' UPDATE Cliente SET Nombre='Fuera Zona' UPDATE Emp_TTotal SET Nombre='Fuera Zona'	UPDATE Persona SET Nombre='Fuera Zona' UPDATE Empleado SET Nombre='Fuera Zona' UPDATE Emp_TParcial SET Nombre='Fuera Zona' UPDATE Cliente SET Nombre='Fuera Zona' UPDATE Emp_TTotal SET Nombre='Fuera Zona'	UPDATE Persona SET Nombre='Fuera Zona' UPDATE Empleado SET Nombre='Fuera Zona' UPDATE Emp_TParcial SET Nombre='Fuera Zona' UPDATE Cliente SET Nombre='Fuera Zona' UPDATE Emp_TTotal SET Nombre='Fuera Zona'	UPDATE Persona SET Nombre='Fuera Zona' UPDATE Empleado SET Nombre='Fuera Zona' UPDATE Emp_TParcial SET Nombre='Fuera Zona' UPDATE Cliente SET Nombre='Fuera Zona' UPDATE Emp_TTotal SET Nombre='Fuera Zona'
UPDATE Emp_TTotal SET Nombre='Fuera ZOna' WHERE Sueldo_Mes=123456789	UPDATE Emp_TTotal SET Nombre='Fuera ZOna' WHERE Sueldo_Mes=123456789	UPDATE Emp_TTotal SET Nombre='Fuera ZOna' WHERE Sueldo_Mes=123456789	UPDATE Emp_TTotal SET Nombre='Fuera ZOna' WHERE Sueldo_Mes=123456789
UPDATE Emp_TTotal SET Nombre='Fuera ZOna', NSS='Que es esto' WHERE Sueldo_Mes=123456789	UPDATE Emp_TTotal SET Nombre='Fuera ZOna', NSS='Que es esto' WHERE Sueldo_Mes=123456789	UPDATE Emp_TTotal SET Nombre='Fuera ZOna', NSS='Que es esto' WHERE Sueldo_Mes=123456789	UPDATE Emp_TTotal SET Nombre='Fuera ZOna', NSS='Que es esto' WHERE Sueldo_Mes=123456789

Tabla 1.3.10 Consultas Update

DELETE

- Delete * from <tabla>
  - Se eliminan todos los registros de las tablas. Al no existir relación entre las tablas para borrar todos los registros se ha de recurrir al uno de varios 'delete'.

- DELETE FROM <*tabla*$_1$>
- DELETE FROM <*tabla*$_2$>
- DELETE FROM <*tabla*$_3$>
- DELETE FROM <*tabla*$_4$>
- DELETE FROM <*tabla*$_5$>

- Delete * from <tabla> where <condición>
  - Se desean borrar todos registros que cumplan cierto criterio.
    - DELETE FROM <*tabla*> WHERE <*campo*>=<*valor*>

La tabla 1.3.11 muestra las consultas Delete realizadas sobre los distintos sistemas gestores de bases de datos

MS Access	Oracle	MySQL	Postgres
DELETE FROM Persona	DELETE FROM Persona	DELETE FROM Persona	DELETE FROM Persona
DELETE FROM Empleado	DELETE FROM Empleado	DELETE FROM Empleado	DELETE FROM Empleado
DELETE FROM Cliente	DELETE FROM Cliente	DELETE FROM Cliente	DELETE FROM Cliente
DELETE FROM Emp_TParcial	DELETE FROM Emp_TParcial	DELETE FROM Emp_TParcial	DELETE FROM Emp_TParcial
DELETE FROM Emp_TTotal	DELETE FROM Emp_TTotal	DELETE FROM Emp_TTotal	DELETE FROM Emp_TTotal
DELETE FROM Emp_TTotal WHERE sueldo_mes=123456789	DELETE FROM Emp_TTotal WHERE sueldo_mes=123456789	DELETE FROM Emp_TTotal WHERE sueldo_mes=123456789	DELETE FROM Emp_TTotal WHERE sueldo_mes=123456789

Tabla 1.3.11 Consultas Delete